VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Server pro podporu výuky teorie her

Supporting server for Games Theory teaching

2013 Tomáš Bařák

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Tomáš Bařák**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Server pro podporu výuky teorie her
Supporting server for Games Theory teaching

Zásady pro vypracování:

Cílem diplomové práce je vytvoření serveru, který bude sloužit k podpoře výuky teorie her. Student si na něm bude moci zahrát vybrané hry, kdy počítač bude uplatňovat optimální strategii, ale také tuto strategii studentovi bude umět zobrazit.
Konkrétní zásady:
- Seznamte se stručně s typy her, které se v teorii her zkoumají
- Podrobně se zaměřte na nestranné kombinatorické hry typu odebíracích her a her Nim
- Navrhněte obecnou softwarovou architekturu serveru tak, aby se daly v budoucnu jednoduše přidávat hry jiného typu
- Vytvořte základní kostru serveru (menu, úvodní stránka apod.)
- Vytvořte část zabývající se odebíracími hrami a hrami typu Nim
- Pro tyto hry bude program umět počítat P a N pozice, Sprague-Grundyovu funkci, optimální tah z dané pozice a toto uživateli zobrazit
- Dále bude možnost tyto hry hrát - 2 uživalé proti sobě nebo uživatel proti počítači, který bude používat optimální strategii

Seznam doporučené odborné literatury:

Thomas S. Ferguson: Game Theory, elektronický výukový text dostupný na adrese http://www.math.ucla.edu/~tom/math167.html (především Part I).

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Martin Kot, Ph.D.**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.*

V Ostravě 2013 .....

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 2013 .....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, zejména vedoucímu této diplomové práce, Ing. Martinu Kotovi Ph.D.

Abstrakt

Tato diplomová práce se zabývá analýzou, návrhem a implementací serveru pro podporu výuky teorie her. Teorie her je odvětví aplikované matematiky, které studuje různé typy her. V rámci této práce se budu zabývat zejména nestrannými kombinatorickými hrami typu odebíracích her a her Nim. V rámci výukové části aplikace student může nastudovat veškerou potřebnou teorii k pochopení principu výše zmíněných her a naučit se, jak zjistit optimální strategii vedoucí k výhře. Na serveru je možné si zahrát několik typů her buď proti počítači, který aplikuje jednu ze zvolených obtížností, nebo ve formě 2 hráčů na jednom počítači. U všech her se také nabízí možnost zobrazit si optimální strategii a instrukce, jak ji zjistit.

Klíčová slova: Java, Vaadin, webová aplikace

Abstract

This diploma thesis deals with analysis, design and implementation of support server for Games Theory teaching. The Game Theory is a branch of applied mathematics that studies various types of games. Within this thesis I will look into details of impartial combinatorial games, especially take-away games and the game of Nim. Students can learn all the theory necessary to understand the principal of above mentioned games along with the optimal strategy leading to the victory of the game, within the educational section of the application. It is possible to play several types of games on the server, either against the computer who applies one of the selected difficulties or in form of two players on one computer. There is also a possibility to display the optimal strategy of all the games together with instructions how to find it out.

Keywords: Java, Vaadin, web application

Seznam použitých zkratek a symbolů

CSS	–	Cascading style sheets
GUI	–	Graphical user interface
HTML	–	Hypertext markup language
IDE	–	Integrated development environment
JAR	–	Java archive
mex	–	Minimal excludant
SG-funkce	–	Sprague Grundyova funkce
UML	–	Unified modelling language
WAR	–	Web application archive
XHTML	–	Extensible hypertext markup language
ZIP	–	Archive file format

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Každý člověk je již od přírody tvor hravý. Počínaje dobou, kdy vývoj smyslů dosahuje takové úrovně, že je dítě schopné vnímat okolní prostředí, začíná také reagovat na různé podněty a hraje si s různými předměty. Poté se začíná i pohybovat, přichází také první objevy v rámci prostoru a člověk se začíná učit. Základní dovednosti jako například rozeznání tvarů nebo barev se většinou děti učí za pomocí nějaké jednoduché hry. Dokonce i Jan Amos Komenský prosazoval učební metody, které zahrnovaly do výukového procesu nenásilné hry, pomocí nichž se člověk snáze a rychleji učí. V dnešní době, kdy je výpočetní technika naprosto běžnou součástí každodenního života, existuje nepřeberné množství různých výukových programů a v mnoha z nich je nedílnou součástí nějaká hra. Cílem této diplomové práce je vývoj výukového serveru, který bude usnadňovat pochopení teoretické i praktické stránky situací v rámci teorie her.

Diplomová práce je rozdělena do dvou částí. První část je teoretická, zabývá se vysvětlením základních pojmů v rámci teorie her, členěním her a konkrétními případy her, které jsou implementovány pro výukový server. U těchto her jsou podrobněji vysvětlena pravidla, herní mechanismy a optimální strategie spolu s principy a algoritmy pro jejich nalezení, které byly použity při implementaci.

Druhá část textu se zaměřuje na praktickou stránku diplomové práce. Rozebírá požadavky na aplikaci, návrh konkrétního řešení výukového serveru a postupy použité při jeho implementaci. Zahrnuje také popis metod testování a nasazení aplikace.

2 Teorie her

2.1 Obecné informace

Existuje nespočet různých her, kterými se lidé baví, at'to jsou hry stolní, karetní, hry hrané na papíře nebo hry počítačové. Teorie her se zabývá také například různými politickými, filosofickými či ekonomickými hrami, ale také studuje soupeření mezi firmami, konflikt mezi managementem a pracovní silou nebo různé právnické spory, a snaží se převést tuto problematiku do podoby her a analyzovat ji. Všechny tyto případy se pak v teorii her odráží v podobě herně-teoretického modelu. Tento model analyzujeme, transformujeme na matematický model a pomocí výpočetní techniky se snažíme nalézt optimální strategii pro veškeré aktéry výše jmenovaných situací (jinak řečeno hráčů her) [1].

Herní modely v rámci teorie her můžeme klasifikovat podle různých kritérií, existují tedy různé typy her. Hry můžeme rozdělovat například podle zisku ze hry, máme tedy hry s nulovým součtem a hry s nenulovým součtem. V rámci her s nulovým součtem je vždy celkový zisk pro všechny hráče roven nule, tzn., že vítěz získává na úkor ostatních hráčů (například hra Go, poker, šachy, dáma,. . .). Naopak pro hry s nenulovým součtem toto pravidlo neplatí, tzn., vítězný hráč tedy může dosáhnout zisku (at'už kladného nebo záporného) zároveň s jiným hráčem (například hra vězňovo dilema). Dále můžeme hry rozdělit podle úrovně poskytnuté informace, a to na hry s dokonalou a nedokonalou informací a hry s úplnou a neúplnou informací. Hry s dokonalou a nedokonalou informací se liší tím, zda hráč zná nebo nezná svou aktuální pozici ve hře, tzn., zná nebo nezná dosavadní průběh hry. V rámci her s úplnou informací hráč zná pravidla hry, neboli zná „Kuhnův strom“ hry, zatímco ve hrách s neúplnou informací jeden nebo oba hráči pravidla nebo část pravidel neznají. Hry s nenulovým součtem se dále dělí na hry kooperativní a nekooperativní. Mezi kooperativní hry řadíme hry, kdy hráči mohou spolupracovat a domluvit se na rozdělení zisku (hry s přenosnou výhrou), a hry, kde zisk není možno dělit (hry s nepřenosnou výhrou). Hry nekooperativní se vyznačují tím, že hráči nespolupracují. Další důležité rozdělení je na nestranné a partyzánské hry. V rámci nestranných her mají možnost oba hráči z jakékoliv pozice vykonat stejný tah, kdežto v partyzánské hře je množina možných tahů z dané pozice pro oba hráče různá. Klasifikaci her ilustruje tabulka 1, str. 8. Existuje několik dalších kritérií pro dělení typů her, těmi se však v této práci dále zabývat nebudu [2].

2.2 Nestranné kombinatorické hry

Hry, na které se v této diplomové práci zaměřím, jsou nestranné kombinatorické hry. Jsou to hry dvou hráčů s dokonalou informací, které končí buď výhrou, nebo prohrou jednoho z hráčů. Tyto hry jsou určeny množinou pozic, zahrnující počáteční (iniciální) a koncovou (terminální) pozici, a hráčem, který je na tahu. Hráči se střídají v tazích, dokud jeden z nich nedosáhne terminální pozice, což je pozice, ze které není možné vykonat žádný tah. Poté je jeden z hráčů označen za vítěze a druhý za poraženého, což záleží na pravidlech hry. Pokud se hraje podle „normálních“ pravidel, vyhrává hráč, který táhl jako

<table>
<tr><td rowspan="2">podle znalosti průběhu hry</td><td>s dokonalou informací</td><td colspan="2" rowspan="2"></td></tr>
<tr><td>s nedokonalou informací</td></tr>
<tr><td rowspan="2">podle znalosti pravidel</td><td>s úplnou informací</td><td colspan="2" rowspan="2"></td></tr>
<tr><td>s neúplnou informací</td></tr>
<tr><td rowspan="2">podle možnosti tahu</td><td>nestranné</td><td colspan="2" rowspan="2"></td></tr>
<tr><td>partyzánské</td></tr>
<tr><td rowspan="3">podle zisku</td><td>s nulovým součtem</td><td colspan="2" rowspan="2"></td></tr>
<tr><td rowspan="2">s nenulovým součtem</td><td rowspan="2">kooperativní</td><td>s přenosnou výhrou</td></tr>
<tr><td>nekooperativní</td><td>s nepřenosnou výhrou</td></tr>
</table>

Tabulka 1: Klasifikace her [2]

poslední, zatímco podle „misére“ (betlových) pravidel poslední hráč na tahu prohrává [2].

2.2.1 Odebírací hry

Jako odebírací hru můžeme označit takovou hru, kde existuje množina nějakých předmětů, které budeme odebírat, v této práci budu uvažovat například hromádku žetonů. Těchto předmětů máme n, kde n je nenulové přirozené číslo. Dále existuje množina S možných tahů, tzn., kolik žetonů je možné odebrat z hromádky, $s \in S$. Hráči se střídají v odebírání žetonů z hromádky a poslední hráč, který má ještě možnost tahu, vyhrává (vezměme v úvahu normální pravidla hry).

Pro určení optimální strategie postupujeme podle jednoduchého algoritmu, jedná se o tzv. zpětnou indukci. Začínáme tedy jakoby od konce hry. Označíme všechny terminální pozice jako P-pozice, neboli pozice, ve kterých vyhrává předchozí hráč (hráč, který se určitým tahem dostal do P-pozice). Pokračujeme v označování pozic tím způsobem, že pokud z dané pozice existuje alespoň jeden tah do P-pozice, označíme pozici jako N-pozici, neboli pozici, ve které vítězí následující hráč. Jako P-pozice potom označíme všechny pozice, ze kterých jakýkoli tah vede do N-pozice. Jako optimální strategii tedy můžeme chápat označení všech pozic ve hře jako P nebo N-pozice a optimální tah vede vždy do P-pozice. Pokud takový tah neexistuje, například hráč 1 začíná v P-pozici, pak hráč 2 použitím optimální strategie musí vždy vyhrát.

V diplomové práci se budu zabývat jednoduchými odebíracími hrami, jejichž odebírací množina se skládá z několika prvků. Také se zaměřím na několik her se zvláštními pravidly, která jsou podrobněji vysvětlena níže [2].

Polovina nebo jeden: První z nich je hra, ve které je možné odebrat maximálně polovinu žetonů. Samozřejmě je nutné odebrat alespoň jeden žeton, terminální pozice hry jsou tedy 0 a 1.

Dim: Dalším speciálním případem odebírací hry je hra Dim, ve které je možné odebrat z hromádky o n prvcích c žetonů, kde c je dělitelem čísla n, včetně čísla 1 a n. Jediná terminální pozice v této hře je 0.

Aliquot: Hra s podobnými pravidly jako Dim se jmenuje Aliquot, liší se pouze tím, že není možné odebrat celou hromádku. Její terminální pozice je 1.

Sudá nebo jeden: Další dvě hry vždy odvozují odebírací množinu pomocí operátoru modulo. V první z nich je možno odebrat počet žetonů, kdy modulo 2 je rovno nule, tedy sudý počet, nebo 1 žeton, pokud jich hra více neobsahuje. Terminální pozice je tedy v této hře 0.

Modulo třemi: Druhá z těchto her umožňuje odebrat násobky 3, pokud se nejedná o celou hromádku, nebo celou hromádku, pokud se počet žetonů modulo 3 rovná 2.

Dalších zajímavých pravidel různých odebíracích her zajisté existuje celá řada a také by asi nebyl problém nějakou novou hru vymyslet, jednalo by se však nejspíše o pravidla odvozená, či podobná výše zmíněným hrám. Pro výukové účely serveru budou tyto hry dostačující.

2.2.2 Hra Nim

Hra Nim může být považována za speciální případ součtové hry. Skládá se totiž z několika odebíracích her, kde každá tato hra má následující pravidla:

- Hra začíná s počtem žetonů n
- Je možno odebrat vždy 1 až k žetonů, kde k je aktuální počet žetonů v této hře
- Může se hrát podle normálních nebo misére pravidel (budu uvažovat normální pravidla)

Výsledná hra Nim se tedy skládá z několika součtových her, které reprezentují buď jednotlivé řádky, nebo sloupce hry. Hráči se opět střídají v tazích a v jednom tahu může hráč odebrat žetony pouze z jedné z odebíracích her (řádku nebo sloupce hry Nim), a to 1 až k žetonů. Vyhrává hráč, který byl na tahu jako poslední, tzn., odebral poslední žeton (žetony).

Pro určení optimální strategie pro aktuální pozici ve hře Nim byl zaveden pojem nim-sum, který bývá označován také jako suma hry Nim.

Definice 2.1 *Nim-sum čísel* $(x_m...x_0)_2$ *a* $(y_m...y_0)_2$ *je* $(z_m...z_0)_2$*, píšeme* $(x_m...x_0)_2 \oplus (y_m...y_0)_2 = (z_m...z_0)_2$*, kde pro všechny* k*,* $z_k = x_k + y_k$*(modulo* 2*), to je* $z_k = 1$*, jestliže* $x_k + y_k = 1$*, jinak* $z_k = 0$*.*

Tuto definici můžeme vysvětlit následujícím způsobem. Suma se určí jako operace xor, neboli exkluzivní logický součet hodnot aktuálního počtu žetonů v jednotlivých odebíracích hrách hry Nim. Jednotlivé hodnoty se převedou do binární soustavy a sčítají se pod sebou. Pokud se ve sloupci nachází sudý počet bitů 1, pak je součet roven 0, jinak 1.

Věta 2.1 *Pozice* (x_1, x_2, x_3) *ve hře Nim je P-pozice tehdy a jen tehdy, když nim-sum těchto komponent je roven nule,* $x_1 \oplus x_2 \oplus x_3 = 0$*.*

Důkaz. Nechť $\mathcal{P}$ je množina pozic hry Nim s nim-sum rovné nule a nechť $\mathcal{N}$ je doplněk této množiny, tedy množina pozic, kdy nim-sum je pozitivní. Zkontrolujme tři podmínky definované v kapitole 2.2.1.

1. Všechny terminální pozice jsou v $\mathcal{P}$. Jediná pozice, která je konečná, je pozice s nulovým počtem chipů ve všech řádcích hry a zároveň $0 \oplus 0 \oplus 0 \oplus .... \oplus 0 = 0$.

2. Z každé pozice z $\mathcal{N}$ existuje tah do pozice z $\mathcal{P}$. Pokud je hodnota sumy Nim nenulová, lze také z její binární reprezentace zjistit, ve kterých řádcích hry Nim existuje optimální tah, tzn., ze kterých her musíme odebrat určitý počet žetonů, abychom se dostali do P-pozice. Tuto informaci poskytuje nejlevější bit nim-sumu, který je roven 1. Hodnoty sčítanců, které v tomto sloupci obsahují rovněž bit 1, jsou pak hry, ve kterých se nachází optimální tah. Počet 1 v tomto sloupci udává, kolik optimálních tahů z této pozice existuje. Pro dosáhnutí P-pozice musíme zmenšit hodnotu v jednom z těchto řádků tak, aby se ve všech sloupcích nacházel sudý počet bitů 1 a výsledná nim-suma se rovnala nule. O kolik jsme toto číslo zmenšili, tolik musíme z daného řádku odebrat žetonů, neboli nalezli jsme optimální tah vedoucí do pozice z $\mathcal{P}$.

3. Každý tah z pozice v $\mathcal{P}$ vede do pozice z $\mathcal{N}$. Jestliže $(x_1, x_2, ...)$ je v $\mathcal{P}$ a x_1 je změněna na $x_1' < x_1$, pak nemůže nastat situace, že $x_1 \oplus x_2 \oplus ... \oplus x_n = x_1' \oplus x_2 \oplus ... \oplus x_n$, protože zákon krácení by implikoval, že $x_1 = x_1'$. Tedy $x_1' \oplus x_2 \oplus ... \oplus x_n \neq 0$, což implikuje, že $(x_1', x_2, ..., x_n)$ je z $\mathcal{N}$.

   Tyto tři vlastnosti ukazují, že $\mathcal{P}$ je množina P-pozic [2].

■

2.2.3 Součtové hry

Pro vytvoření součtové hry je třeba několik samostatných odebíracích her, které mají svá pravidla. Pro příklad vezměme v úvahu hru o 15 žetonech, kde je možno odebrat 1, 2, 4 nebo 7 žetonů, další hra začíná na 19 žetonech a odebírací množinu tvoří prvky 1, 3, 4 a poslední hra je typu Nim a obsahuje 7 žetonů, tedy je možno odebrat 1 až 7 žetonů. Tyto 3 hry tvoří dohromady součtovou hru, ve které se opět střídají 2 hráči. Tah jednoho hráče spočívá v odebrání několika žetonů z jedné z her, avšak důležité je z každé hry odebírat podle pravidel této konkrétní hry. Hráč, který odebere poslední žeton, vyhrál, druhý hráč prohrál. Součtovou hru můžeme zobecnit tak, že máme n her s vlastními pravidly (odebírací množina) a tyto hry jsou "podhrami" výsledné součtové hry.

K tomu, aby bylo možné najít optimální strategii obecné součtové hry, je nutné použít Sprague-Grundyovu (SG) funkci jednotlivých her. Tato funkce přiřazuje přirozená čísla (včetně nuly) jednotlivým pozicím ve hře podle následujícího funkčního předpisu:

$$g(x) = \min\{n \geq 0 : n \neq g(y) \text{ pro } y \in F(x)\}$$

kde F je funkce, která každému x přiřadí množinu následníků (do jaké pozice je možné se z dané pozice x odebráním n žetonů dostat). Sprague-Grundyova funkce tedy přiřazuje

nejmenší nezáporné číslo, které se nenachází v množině hodnot Sprague-Grundyovy funkce následníků x. Můžeme definovat funkci mex (minimal excludant) z množiny přirozených čísel jako nejmenší přirozené číslo, které se nenachází v této množině, pak se vztah zjednoduší na:

$$g(x) = \text{mex}\{g(y) : y \in F(x)\}$$

Tato funkce je definována rekurzivně, přístup je tedy podobný jako u zpětné indukce při určování P a N-pozic. Postupujeme tedy od terminální pozice, která má hodnotu Sprague-Grundyovy funkce rovnu 0 (v množině $F(x)$ se nenachází žádný prvek, nejmenší přirozené číslo je tedy 0). Další hodnoty pak závisí na odebírací množině příslušné hry. Po určení hodnot SG-funkce všech pozic ve hře můžeme zjistit také, které z pozic jsou typu N a P a to tak, že pro hodnotu SG-funkce rovnu 0 je pozice typu P a pro všechny ostatní je to N-pozice.

Jakmile známe hodnoty SG-funkce pro všechny hry, ze kterých se součtová hra skládá, můžeme určit optimální strategii. Princip je analogický k určování hodnoty nim-sum u hry Nim, jelikož tato hra je speciálním případem součtové hry. Obecně tedy pro součtovou hru o n podhrách vybereme z každé této hry hodnotu SG-funkce v k-té pozici hry, kde k je aktuální počet žetonů v této hře. Provedeme opět exkluzivní logický součet těchto hodnot a podle výsledku zjistíme, v jaké pozici se součtová hra nachází. Pokud je tato suma rovna 0, jedná se o P-pozici, jinak je hra v N-pozici.

Optimální tah poznáme buď podle nejlevějšího bitu v sumě rovnému 1 (ve hrách, které mají nejlevější bit binární reprezentace hodnoty SG-funkce roven 1, nalezneme, kolik žetonů je nutno odebrat.), nebo je také možné, že nastane situace, kdy se optimální tah nachází v jiné hře. V některých případech tomu tak není. Je to způsobeno tím, že v rámci součtových her existují odebírací hry, které mají hodnoty Sprague-Grundyho funkce nerostoucí (respektive neklesající). Poté se může stát, že odebráním určitého počtu žetonů se z pozice, kdy je hodnota SG-funkce nižší, dostaneme do pozice, kdy je tato hodnota vyšší, což může způsobit vyrovnání sumy celkové hry na 0. Je tedy nutné uvažovat všechny možnosti, kdy se z dané N-pozice můžeme dostat do P-pozice, a zkontrolovat, jestli takový tah je z dané pozice možný [2].

3 Analýza požadavků

Podstatou diplomové práce bylo zkonstruovat server pro podporu výuky teorie her, proto se tato část textu bude zaměřovat na všechny fáze jeho vývoje. Dalo by se říci, že vývojový proces probíhal v iteracích. Hlavní požadavky na aplikaci byly sice dány v rámci zadání diplomové práce, ale při konzultaci s vedoucím práce došlo vždy k jejich přesnější specifikaci. Analýza a návrh řešení probíhal většinou zároveň se zadáním požadavků a po implementaci a zevrubném testování funkčnosti se iterace opakovala.

Všechny požadavky se dělí na funkční a nefunkční. Funkční požadavky zahrnují veškeré požadavky na funkcionalitu aplikace, zatímco nefunkční požadavky obsahují jak požadavky na výkonnost serveru, tak jeho grafické zpracování a design.

3.1 Funkční požadavky

3.1.1 Základní kostra serveru

Pro chod funkční aplikace je třeba, aby byla rozdělena prostřednictvím uživatelského rozhraní na logické celky, které představují jednotlivé stránky serveru. Mezi nimi je nutné se nějakým způsobem přepínat. Je tedy důležité navrhnout ovládání serveru, tak aby bylo uživatelsky přívětivé a intuitivní.

3.1.2 Odebírací hry

Prostřednictvím serveru bude možné si zahrát několik typů jednoduchých odebíracích her. Aplikace nabídne několik úrovní obtížnosti v případě hry proti počítači, ale také hru dvou hráčů na jednom stroji. Uživatelské rozhraní umožní volbu velikosti množiny předmětů, ze které se bude odebírat, a rovněž množinu možných tahů k odebrání. Aplikace zobrazí optimální strategii pro danou hru, tedy P a N-pozice, Sprague-Grundyovu funkci a optimální tah z dané pozice. Všechny důležité pojmy, výpočty a algoritmy budou vysvětleny.

3.1.3 Hra Nim

Server umožní zahrát si hru Nim, a to buď proti počítači, který svou taktiku používá podle zvolené úrovně obtížnosti, nebo proti lidskému soupeři. V rámci uživatelského rozhraní bude možné nastavit, z kolika řádků se bude hra Nim skládat, případně kolik předmětů bude maximálně řádek obsahovat. Aplikace umožní zobrazení výpočtu nim-sumy, optimálního tahu (optimálních tahů) z dané pozice a dalších informací důležitých k porozumění herního mechanismu. Výuku usnadní doplňující text, který vysvětlí veškeré pojmy a principy.

3.1.4 Součtové hry

V rámci aplikace si uživatel bude moci zahrát libovolnou součtovou hru, uživatelské rozhraní umožní modifikovat její vlastnosti. Bude možné konkrétně nastavovat jednotlivé

odebírací hry, ze kterých se výsledná součtová hra bude skládát. Hru bude opět možno hrát proti počítači, který bude uplatňovat strategii podle zvolené obtížnosti, nebo proti druhému hráči na stejném počítači. Server zobrazí strategii pro výuku výpočtu P, N-pozic, Sprague-Grundyovy funkce jednotlivých her a také sumy her. Dále bude umožněno zobrazení optimálního tahu z dané pozice a doplňující informace ke hře.

3.1.5 Teorie

Server umožní zobrazení teorie, kterou je nutné nastudovat pro pochopení výše zmíněných typů her a nalezení optimální strategie k jejich výhře. Pro uživatele, kteří budou oprávněni ke změně této teorie (vyučující, cvičící, ...), bude umožněna její editace prostřednictvím uživatelského rozhraní aplikace.

3.1.6 Nápověda

Uživatel si bude moci zobrazit nápovědu k aplikaci v podobě uživatelské příručky. Tento dokument bude zobrazen na samostatné stránce v rámci aplikace. Budou popsány jednotlivé stránky aplikace spolu s akcemi, které je možné provést. Tyto akce budou vysvětleny a příručka zprostředkuje návod k tomu, jak je provádět.

3.2 Nefunkční požadavky

3.2.1 Design aplikace

Výukový server bude splňovat požadavky klasické webové aplikace, vzhled grafického uživatelského rozhraní by měl být snadno ovladatelný, logicky uspořádaný a intuitivní. Uživateli by neměla činit problémy orientace v aplikaci, případně nastavení a ovládání jednotlivých typů her.

3.2.2 Architektura systému

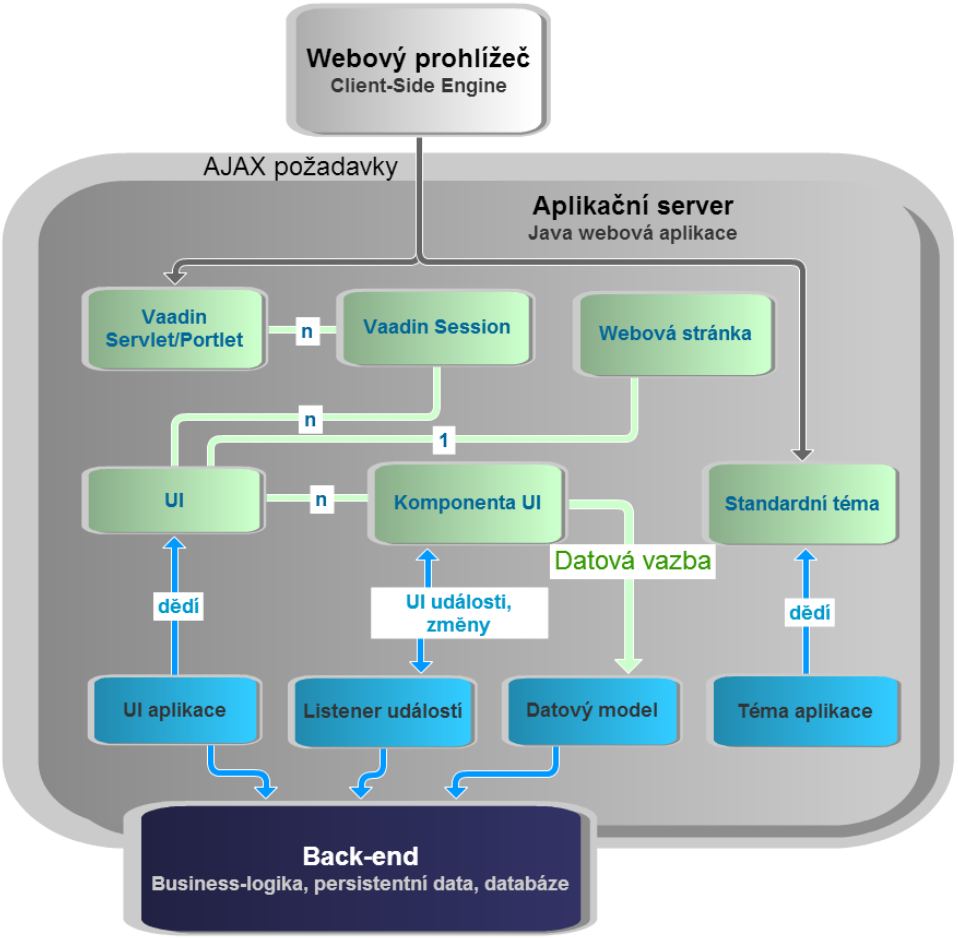
Architektura výsledného systému umožní jednoduché přidávání nových her jiného typu v rámci stávajícího serveru. Aplikace bude optimalizovaná, s přiměřenými hardwarovými nároky. Nasazení systému bude provedeno na server s podporou Javy jako webová aplikace.

4 Návrh

Zadání diplomové práce specifikuje základní požadavky na vzhled a funkčnost aplikace. Podle něj jsem tedy vytvořil jednoduchý návrh uživatelského rozhraní, struktury serveru a zevrubný nástin grafického zpracování jednotlivých her.

4.1 Architektura

Server pro podporu výuky teorie her je ve skutečnosti webová aplikace, jejíž architektonická struktura se skládá z několika komponent. Protože jsem se rozhodl server vyvíjet v programovacím jazyce Java za pomocí frameworku Vaadin [4], architektura aplikace se odvíjí od základní struktury serverových aplikací založené na tomto frameworku (viz Obrázek 1, str. 14) [3]. Java aplikace vytvořená ve Vaadinu běží jako Java servlet za pomocí



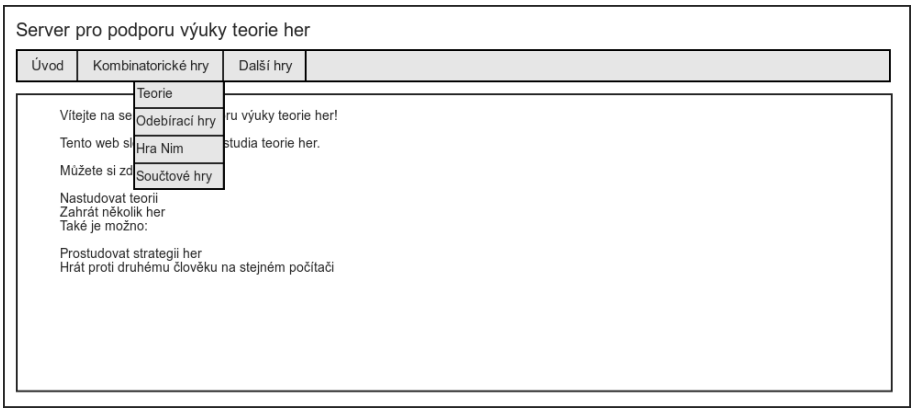
Obrázek 1: Architektura serverové webové aplikace založené na frameworku Vaadin

servlet kontejneru. Aplikace se skládá z mnoha server-side komponent, které dohromady tvoří grafické uživatelské rozhraní. Tyto komponenty interagují pomocí listenerů s aplikační logikou, tedy business třídami zastřešujícími funkční podstatu aplikace. Design aplikace zajišťuje její téma (v diagramu „Téma aplikace“), které je odvozeno od tématu standardního. V tomto tématu je možné upravit vzhled pomocí CSS stylů, nebo například doplnit aplikaci o obrázky, nebo jiná multimédia prostřednictvím zdrojů. Může se jednat o zdroje poskytnuté aplikací, serverem, nebo externí zdroje. Původní návrh aplikace zahrnoval přepínání grafických témat za běhu, vzhledem k obtížné optimalizaci rozmístění komponent, zarovnání, zobrazení her a dalším problémům, jsem však od tohoto systému byl při implementaci nucen upustit.

Návrh třídní struktury reprezentace jednotlivých komponent, které tvoří aplikaci, probíhal spíše intuitivně, jelikož jsem již v minulosti nějaké zkušenosti s vývojem pomocí frameworku Vaadin měl. Základní myšlenka byla v oddělení aplikační logiky od GUI, což se do jisté míry povedlo. Komunikace těchto dvou hlavních komponent probíhá prostřednictvím systému akcí a stavů, neboli výše zmíněných listenerů.

4.2 Grafické uživatelské rozhraní

Prvotní nástin grafického zpracování aplikace jsem vytvořil pomocí nástroje pro tvorbu wireframe webu. Tento návrh je velice jednoduchý a zobrazuje pouze základní vzhled serveru a ovládací prvky pro orientaci na něm (viz Obrázek 2, str. 15). Počáteční myšlenka designu aplikace byla zachována, upravil jsem pouze detaily pomocí grafického frameworku. Pro jednoduchost ovládání jsem použil hlavní rozbalovací menu, pomocí kterého je možné pohybovat se mezi jednotlivými stránkami serveru.



Obrázek 2: Návrh úvodní obrazovky, uživatelského rozhraní

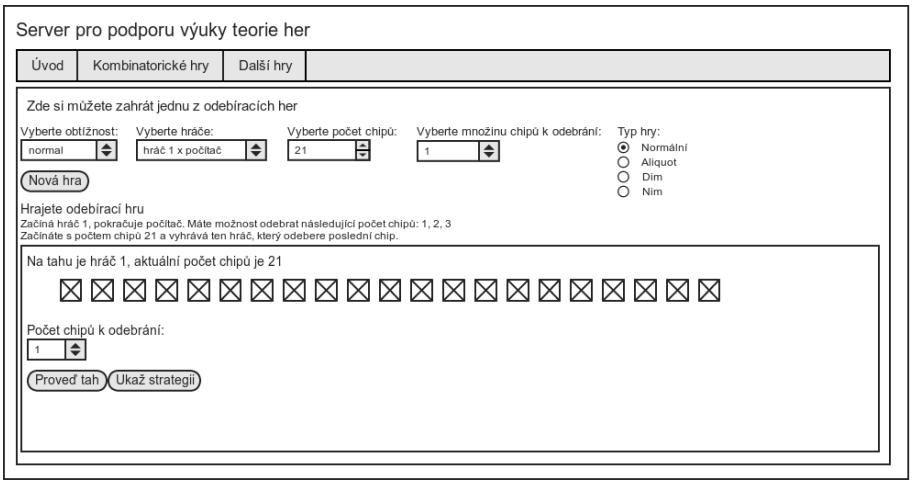
4.3 Teorie

Návrh zobrazení příslušné teorie týkající se nestranných kombinatorických her nebyl složitý, jednalo se pouze o zobrazení formátovaného textu. K tomuto prvotnímu poža-

davku se později přidala možnost upravit tento text oprávněným uživatelem. Editace je umožněna po přihlášení uživatele za pomocí jednoduchého login formuláře.

4.4 Zpracování her

Hlavním cílem serveru je možnost zahrát si různé typy her a naučit se optimální strategii pro jejich výhru. Proto bylo nutné navrhnout grafické zpracování odebíracích her tak, aby do zobrazení hry bylo možné zakomponovat také informace o optimální strategii a také algoritmy a doporučení, pomocí kterých je možné se k této strategii dopracovat. Samozřejmě jsem musel vzít v úvahu návrh ovládacích prvků jak pro nastavení hry, její spuštění, tak samotné ovládání. Snažil jsem se o jednoduchost, aby byla zdůrazněna funkčnost aplikace. Jako prvky odebíracích her jsem zvolil žetony (chipy), které jsou používané v hazardních hrách jako poker, ruleta, atd. Inspirací mi byl výukový text, ze kterého jsem čerpal při studiu teorie her. Existují různé možnosti zobrazení množiny žetonů, ze kterých se v průběhu hry odebírá, já jsem zvolil uspořádání do řádku. V prvotní fázi návrhu jsem zamýšlel implementovat odebírání žetonů pomocí kliknutí na něj. Později jsem však došel k závěru, že tato metoda je sice intuitivní a jednoduchá, přináší ale v jistých ohledech nepřehlednost a pro výukové účely jsem zvolil metodu výběru počtu žetonů k odebrání pomocí rozbalovacího seznamu. Výsledný návrh se skládá ze dvou částí, a to nastavení hry a její zobrazení, které dále zahrnuje ovládací prvky a také komponenty pro zobrazení výukových informací ke hře (viz Obrázek 3, str. 16). Tento



Obrázek 3: Návrh zobrazení odebíracích her

wireframe byl postupně přizpůsobován nové funkčnosti aplikace a zobrazení dalších typů her. Pro součtové hry a hru Nim je například použito několik komponent grafického rozhraní odvozených od jednoduchých odebíracích her.

4.5 Uživatelská příručka

Návrh zobrazení uživatelské příručky nebyl příliš složitý, snažil jsem se o jednoduchost, čemuž odpovídá rozdělení do 4 sekcí podle jednotlivých podmenu kombinatorických her. V úvahu přicházelo klasické zobrazení nápovědy pomocí obsahu, nebo rozdělení pomocí rejstříku, což mi přišlo zbytečně komplikované. Využil jsem tudíž jednoduché zobrazení členěné pomocí záložek, kde v každé záložce se nachází stránka o příslušné sekci. Tato stránka je textového charakteru doplněna ilustrujícími obrázky.

5 Implementace

Jak již bylo zmíněno, implementace aplikace probíhala v iteracích. Před začátkem implementační části iterace předcházelo upřesnění detailů v podobě diskuse ohledně návrhu zpracování požadavků. Systém je naimplementován v jazyce Java s využitím frameworku Vaadin. Používal jsem vývojové prostředí Eclipse, do kterého je tento framework jednoduše integrován pomocí pluginu. Pro spuštění webové aplikace sloužil virtuální server Tomcat, který je rovněž zahrnut ve vývojovém prostředí. Pomocí tohoto serveru jsem pak byl schopen ladit aplikaci ve webovém prohlížeči. Snažil jsem se optimalizovat výsledný systém pro všechny nejrozšířenější internetové prohlížeče jako je Google Chrome, Internet Explorer a Mozilla Firefox.

5.1 Vaadin

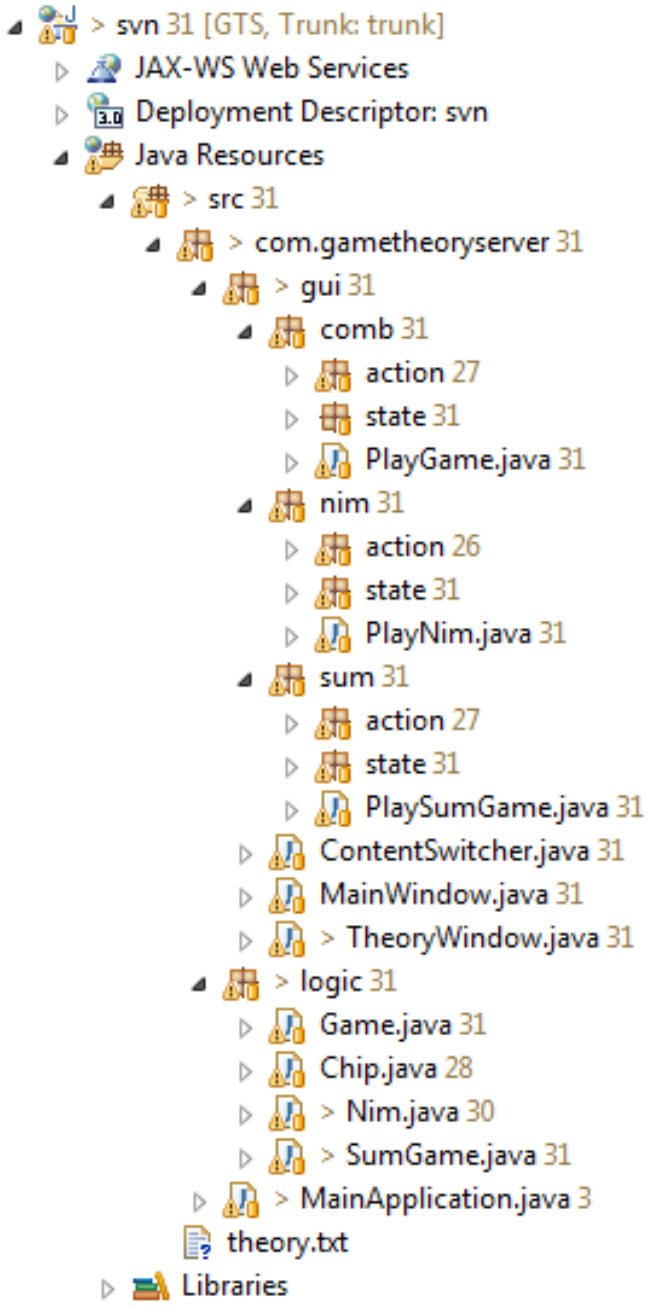
Vaadin je framework usnadňující vývoj webových aplikací v jazyce Java. Umožňuje vývoj aplikací jak ze strany klienta, tak serveru, neboli tzv. server-side a client-side přístup. Implementace pomocí server-side modelu se více podobá vývoji klasických Java desktop aplikací, zatímco pokud se zaměříme na vývoj ze strany klienta, je třeba zvolit přístup webového programování, tedy používat různé webové technologie (např. HTML, CSS, javascript, ...). V mém případě byla jasná volba vývoje na straně serveru.

Implementace pomocí frameworku Vaadin je založena na komponentách, dalo by se tedy říci, že se jedná o komponentní framework. Uživatelské rozhraní aplikace se skládá z jednotlivých komponent, které většinou tvoří layouty jednotlivých stránek. Každá komponenta má nespočet nastavitelných vlastností, díky nimž lze jednoduše upravovat jak její vzhled, tak do určité míry i chování. Celkové grafické zpracování je také možné doladit pomocí CSS stylů v rámci tématu aplikace. Reakce na podněty je řešena prostřednictvím listenerů, tedy GUI je pomocí nich propojeno s logikou aplikace. Veškeré součásti Vaadinu jsou náležitě zdokumentovány, a to buď ve formě Knihy o Vaadinu [3], nebo přímo na webu Vaadinu [4], kde se nachází přehledná dokumentace spolu s ukázkami (tzv. sampler), ale také přímo API Vaadinu s dokumentací všech použitých tříd.

V IDE Eclipse existuje designer pro Vaadin, díky němuž lze vcelku rychle a efektivně dosáhnout uspokojivého výsledku v podobě GUI, má však i své nedostatky. Komponenty není možné náležitě customizovat, návrh uspořádání do layoutů je automatický a jelikož se jedná o samostatnou aplikaci vyvinutou pomocí frameworku Vaadin, práce pomocí tohoto pluginu není příliš plynulá. Designer také není úplně optimalizovaný, takže se často objevují nevysvětlitelné chyby. Celkově se mi práce v tomto editoru nezamlouvala, proto jsem jej nepoužíval.

5.2 Aplikace

Tvorba webové aplikace pomocí frameworku Vaadin je jednoduchá. V prostředí Eclipse je možné automaticky vygenerovat projekt, který obsahuje všechny potřebné náležitosti a je správně nastaven pro spuštění. Struktura projektu byla rozdělena do dvou hlavních

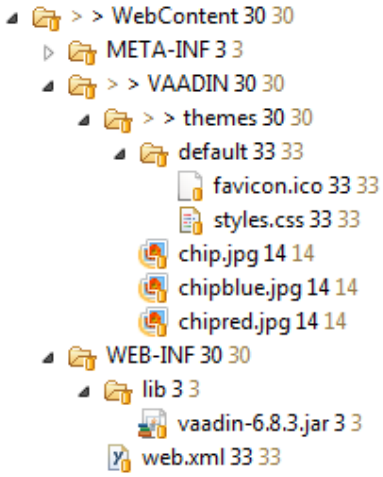


Obrázek 4: Struktura projektu

částí, a to balíku `gui`, který obsahuje všechny třídy související s GUI aplikace, a balíku `logic`, obsahující třídy implementující aplikační logiku (viz Obrázek 4, str. 19).

5.2.1 Projekt

Každý projekt vytvořený pomocí frameworku Vaadin obsahuje mimo výše uvedé Java zdrojové soubory ještě složku WebContent, která se dělí se na tři podadresáře. První z nich je META-INF, obsahující meta informace o aplikaci. Další složka je pojmenována VAADIN a obsahuje grafická témata aplikace spolu se všemi zdroji, které jsou použity. Posledním podadresářem je WEB-INF, který zahrnuje další složku lib s aktuální použitou verzí frameworku Vaadin a nachází se v něm také soubor web.xml důležitý pro nastavení způsobu běhu aplikace a mapování servletu (viz Obrázek 5, str. 20).



Obrázek 5: Struktura adresáře WebContent

5.2.2 MainWindow

Podstatou aplikace je třída `MainApplication`, která dědí ze třídy `Application`. Díky této třídě je aplikaci přiděleno okno, které je realizováno instancí třídy `MainWindow`. Konkrétní objekt typu `MainWindow` se skládá z komponent tvořících layouty, které odpovídají horizontálnímu, respektive vertikálnímu členění stránky. Horizontální a vertikální layouty dohromady vytvářejí vzhled okna.

V této třídě je vytvořen celkový vzhled aplikace pomocí hlavního layoutu obsahujícího několik komponent, které jsou uspořádány vertikálně. První z nich je název serveru realizovaný komponentou typu `Label`. Název aplikace následuje hlavní menu, které tvoří komponenta `MenuBar` a skládá se z jednotlivých položek typu `MenuItem`. Třetí a poslední komponentou hlavního okna aplikace je `contentLayout`, který zobrazuje aktuální obsah okna. Tato část stránky je překreslována při přepínání obsahu pomocí hlavního menu aplikace.

5.2.3 ContentSwitcher

Přepínání mezi jednotlivými stránkami aplikace realizuje třída `ContentSwitcher`, která reaguje na podněty zaslané prostřednictvím hlavního menu aplikace a nahrazuje stávající kontext obsahem požadovaného odkazu, na který bylo v rámci menu kliknuto. Protože všechny ostatní třídy GUI realizující jednotlivé stránky aplikace dědí ze třídy `VerticalLayout`, je přepnutí kontextu umožněno jednoduchou výměnou tohoto layoutu v rámci hlavního okna aplikace. Obrázek 6 na straně 21 ilustruje grafické zpracování aplikace, její hlavní stránky a menu.

Obrázek 6: Úvodní obrazovka, hlavní menu aplikace

5.3 Teorie

Stránka obsahující teorii k prostudování má základ ve třídě `TheoryWindow`. Implementace zobrazení textu je jednoduchá, pomocí komponenty `Label` je zobrazen formátovaný text, který se načítá ze zdrojového souboru uloženého na serveru. Tato komoponenta je následně zarovnána do `VerticalLayoutu`. Soubor má formát XHTML pro zachování formátování textu.

Prvotní myšlenka při implementaci vstupně-výstupních metod zahrnovala použití klasického přístupu k souboru přes datové toky (třídy `FileInputStream`, `Scanner`, `FileOutputStream`, `OutputStreamWriter` k načtení souboru a zapsání do něj). Takto vznikly metody `readFile`, `writeFile` pro čtení a zápis. Později jsem si však uvědomil, že bude zapotřebí přístupových práv k tomu, aby nějaký soubor na serveru mohl existovat. Implementoval jsem tedy rovněž metodu, která načte soubor přímo z kontextu aplikace, čímž zajistím, že nebude třeba přístupových práv na serveru. Realizaci tohoto přistupu odpovídají metody `readText`, `writeText`.

Obrázek 7: Editace textu teorie

Později bylo nutné přidat funkční přihlášení pro uživatele, který požaduje editaci textu, což jsem realizoval pomocí modálního vyskakovacího okna a jednoduchého přihlašovacího formuláře v něm. Úprava textu je možná díky komponentě `RichTextArea`, která umožňuje potřebné základní formátování (viz Obrázek 7, str. 22). Po stisknutí tlačítka „Použít" je pak text uložen do výše zmíněného souboru na serveru. Pokud je uživatel přihlášen, má rovněž logicky možnost se odhlásit prostřednictvím tlačítka „logout".

5.4 Logika Aplikace

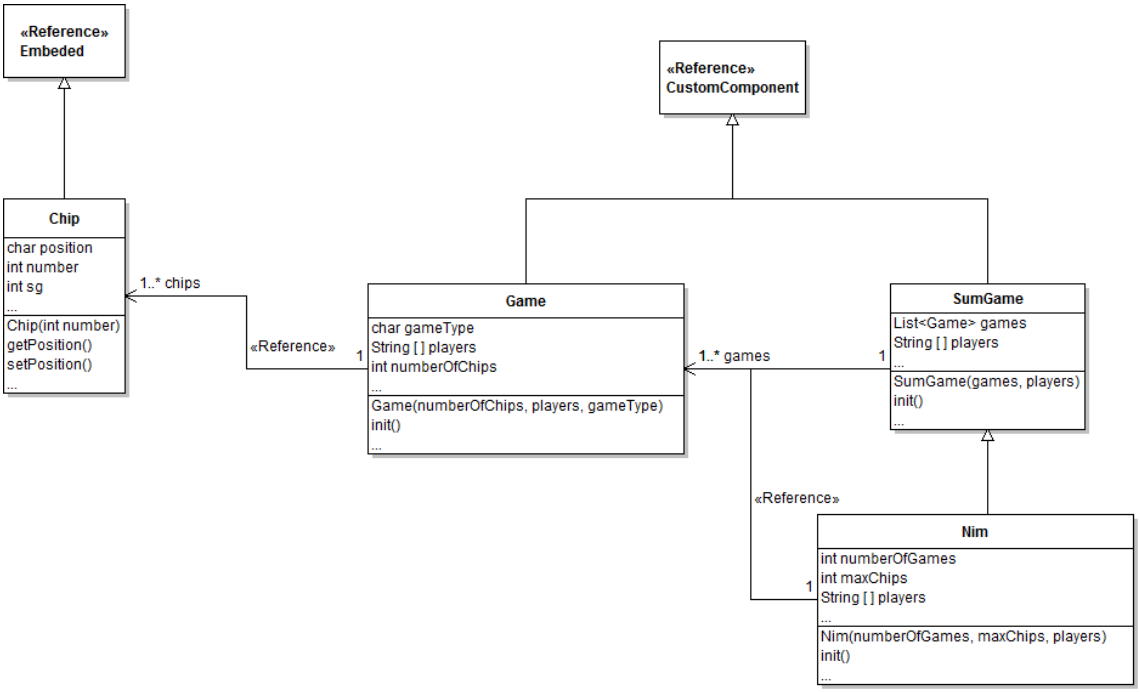
Pro oddělení tříd implementujících logickou podstatu aplikace od tříd GUI byl v hierarchické struktuře projektu vytvořen balík `logic`. Logiku aplikace reprezentují třídy `Chip`, `Game`, `Nim` a `SumGame`. S ohledem na návrh a reálné fungování kombinatorických her byl původní záměr takový, že třída `Game` bude tvořena objekty typu `Chip` a třídy `Nim` a `SumGame` budou tvořeny instancemi třídy `Game` (viz Obrázek 8, str. 23).

Vytvoření instance všech těchto tříd probíhá pomocí konstruktoru, ve kterém dojde také k inicializaci třídy pomocí metody `init`. Tato metoda připraví všechny potřebné privátní proměnné tříd, komponenty k jejímu zobrazení a přichystá hru k použití.

5.4.1 Chip

Pro reprezentaci žetonů ve hře byla navržena třída `Chip`, která zapouzdřuje všechny atributy a metody konkrétního chipu. Instance této třídy si tedy v sobě drží informace o čísle žetonu ve hře, hodnotu SG-funkce a pozici, kterou tento chip představuje (P nebo N pozice). K těmto atributům je možné přistupovat pomocí veřejných (public) metod, které jejich hodnotu mohou buď zjistit (gettery), nebo tuto hodnotu mohou nastavit (settery).

Protože je třída `Chip` potomkem třídy `Embedded`, což je komponenta, pomocí které se zobrazují obrázky a další multimédia, je nutné každé instanci této třídy přiřadit její zdroj,



Obrázek 8: Třídní diagram balíku `logic`

tedy obrázek. Tato procedura je realizována pomocí metody setSource třídy `Embedded`, která přiřadí objektu požadovaný zdroj. Použitý zdrojový obrázek je obsažen v grafickém tématu aplikace a je přiřazen jako typ `ThemeResource`.

5.4.2 Game

Obrázek 9: Zobrazení žetonů ve hře

Implementace podstaty odebírací hry byla provedena pomocí třídy `Game`, která zahrnuje veškerá data a funkce hry. Objekt, který je vytvořen konstrukcí této třídy obsahuje všechny chipy konkrétní hry, informace o této hře v podobě instančních proměnných a veškeré metody umožňující hraní hry a zobrazení informací o optimální strategii. Třída `Game` dědí ze třídy `CustomComponent`, je to tedy komponenta uživatelského rozhraní rozšířená o funkčnost. Tento způsob implementace jsem zvolil z důvodu pozdějšího použití této komponenty ve hře Nim a v součtových hrách, kde se výsledné hry ve skutečnosti skládají z několika jednoduchých odebíracích her. Základem třídy `Game` je množina chipů, které hru tvoří, jsou to konkrétní objekty třídy `Chip`. Tyto chipy jsou naskládány do horizontálního layoutu pro jejich jednoduché zobrazení v řádku (viz Obrázek 9, str. 23). Zároveň byla pro udržení objektů žetonů v rámci třídy `Game` použita kolekce typu `List`

kvůli jednoduché manipulaci (odebírání chipů při hře, zjišťování hodnot SG-funkce, zda se jedná o P nebo N-pozici, atd.).

Pro výukové účely bylo nutné naimplementovat metody třídy `Game`, které budou počítat hodnoty Sprague-Grundyovy funkce a zda se jedná o P nebo N-pozici, pro jednotlivé chipy ve hře. Konkrétně se jedná o metodu `setSGfunction`, která každému chipu přiřadí podle jeho pozice ve hře hodnotu SG-funkce za pomocí metody `mex` (viz Výpis 1, str. 24).

```
public static int mex(HashSet<Integer> set) {
    if (set.isEmpty())
        return 0;
    int max = Collections.max(set), i = 0;
    for (i = 0; i <= max; i++) {
        if (!set.contains(i))
            break;
    }
    return i;
}
```

Výpis 1: Implementace pomocné funkce mex

Původně byla tato metoda funkční pouze pro jednoduché odebírací hry, později bylo nutné přidat pomocí příkazu `switch` varianty výpočtu hodnot SG-funkce pro speciální typy her, které se vyznačují odlišnými pravidly. Výpočet vždy závisí na množině hodnot, které je možné odebrat ze hry v dané pozici.

Metoda pro určení pozic ve hře (`setPNpositions`), tedy zda se jedná o P nebo N-pozici, prochází jednotlivé chipy a pozici přiřazuje podle hodnoty SG-funkce. Pokud je hodnota nulová, pak se jedná o P pozici, jinak je daný chip N pozicí. Tyto informace jsou pak pomocí metody `getStrategyTable` uspořádány do tabulky, kterou je posléze možno zobrazit.

Další metoda počítá obecný vzorec pro určení SG-funkce za pomocí funkce pro zjištění opakování v rámci posloupnosti hodnot SG-funkce v případě jednoduché odebírací hry. Pro další typy her pouze vrátí příslušný vzorec jako formátovaný text v řetězci. Tento vzorec je pak součástí zobrazené nápovědy sloužící pro výukové účely.

Poslední metoda, která souvisí se zobrazením informací podstatných pro výuku je `getBestMove`, která počítá optimální tah z dané pozice. Tato metoda je využita jak pro realizaci tahu počítače, tak při zobrazení optimálního tahu v rámci informací o strategii uvnitř metody `getBestMoveString`. Kvůli implementaci různých typů odebíracích her bylo nutné zajistit, aby tato metoda zohlednila několik možných tahů z dané pozice, její návratová hodnota je tedy typu `SortedSet<Integer>`. Tato generická kolekce zajišťuje, že hodnoty typu `Integer` budou seřazeny.

Pro funkčnost hraní her byly naimplementovány další metody třídy `Game`, a sice `isMovePossible`, `randMove`, `removeChips`, `computerPlay`, `countTakeAways` a `repaint`. Názvy těchto metod jsou do jisté míry vypovídající. Metoda `isMovePossible` určuje, zda je tah z dané pozice uskutečnitelný za pomocí jednoduchého výpočtu, zda odebraný počet chipů koresponduje s jednou z hodnot z množiny chipů k odebrání. Tuto množinu je kvůli některým typům her, kde se v průběhu hry mění její prvky, nutno

obnovovat, k čemuž slouží metoda `countTakeAways` použitá uvnitř metody `repaint`. `RandMove` vrací náhodný tah, který je zprostředkován funkcí `Collections.shuffle` provedenou nad odebírací množinou a následným vybráním prvního prvku. Pomocí metody `removeChips` jsou odebrány žetony z aktuální pozice ve hře.

```
public int computerPlay(String difficulty ) {
    int numberOfChips = chips.size();
    int num = 0;
    SortedSet<Integer> moves = getBestMove();
    Chip chipToPlay = null;
    //pokud je nalezen optimalni tah
    if (moves.size() > 0) {
        //vyberu z mnoziny optimalnich tahu jeden nahodny
        List<Integer> asList = new ArrayList<Integer>(moves);
        Collections.shuffle(asList);
        num = asList.get(0);
        chipToPlay = chips.get(numberOfChips - num);
    } else {
        //jinak provedu nahodny tah
        num = randMove();
        chipToPlay = chips.get(numberOfChips - num);
    }
    //tah pri obtiznosti easy
    if (difficulty.equals("easy")) {
        if (computerMove % 5 != 0) {
            num = randMove();
            chipToPlay = chips.get(numberOfChips - num);
        }
    //tah pri obtiznosti normal
    } else if (difficulty.equals("normal")) {
        if (computerMove % 3 != 0) {
            num = randMove();
            chipToPlay = chips.get(numberOfChips - num);
        }
    }
    //odeberu pozadovane chipy
    removeChips(chipToPlay);
    computerMove++;
    //vratim pocet odebranych chipu
    return num;
}
```

Výpis 2: Metoda pro tah počítače

Metoda `computerPlay` zprostředkovává tah počítače v závislosti na parametru `difficulty`, tedy obtížnosti hry. Za pomocí privátní proměnné třídy typu `int`, která je v každém tahu počítače inkrementována, je při obtížnosti normal každý třetí tah počítače realizován pomocí optimální strategie a při obtížnosti easy je tomu tak u každého pátého tahu (viz Výpis 2, str. 25). Metoda `repaint` překresluje hru, tzn., obnoví informace o hře (který hráč je aktuálně na tahu, počet chipů ve hře, odebírací množina) a její vzhled (hra bude zobrazena po odebrání příslušného počtu chipů). Pomocí této metody se také

kontroluje, zda je hra již ukončena a tato skutečnost je zaznamenána v podobě privátní proměnné `finished` typu `boolean`.

5.4.3 SumGame

Hra Nim je teoreticky vzato konkrétním případem součtové hry, kde všechny podhry se hrají podle pravidel hry Nim. Z objektově orientovaného pohledu je tedy třída `SumGame` rodičem třídy `Nim`, sama však dědí opět ze třídy `CustomComponent`, podobně jako třída `Game`. Z hlediska frameworku Vaadin se tedy opět jedná o komponentu uživatelského rozhraní. Třída `SumGame` se opět skládá z instancí třídy `Game`, které jsou řazeny do seznamu a graficky uspořádány do vertikálního layoutu v podobě jeho jednotlivých řádků. Pro přehlednost jsou v lichých řádcích použity černé žetony a v sudých modré.

Jelikož princip hraní součtových her a hry Nim je podobný, co se týká odebírání žetonů, i implementace metod zprostředkovávající tuto funkčnost v příslušných třídách je podobná. Rozdíl však tvoří pravidla, která jsou v jednotlivých podhrách součtové hry specifická, proto je odlišný výpočet optimálního tahu pro součtové hry. Na rozdíl od hry Nim, kde hodnoty SG-funkce jsou ve všech hrách stejné, tedy pro každý žeton se hodnota SG-funkce rovná jeho pořadí ve hře, pro součtové hry je nutné hodnoty SG-funkce vypočítat pro každou podhru zvlášť. Pro různá pravidla her je taky možné, že existuje více optimálních tahů, které se nacházejí v rámci jednoho řádku součtové hry, z některých pozic ve hře. Proto metoda `getChipsArrayToRemove` operuje s dvourozměrným polem pro uchování optimálních tahů, jelikož v každém řádku hry může existovat několik takových tahů. Tato metoda vrací právě dvourozměrné pole, kde prvním rozměrem jsou čísla jednotlivých podher v rámci součtové hry a druhým rozměrem jsou počty chipů které je nutné v těchto hrách odebrat pro realizaci optimálního tahu. Následující úryvek kódu ilustruje implementaci této metody.

```
public int [][] getChipsArrayToRemove() {
    int [] sgvalues = new int[games.size()];

    int i = 0;
    int takes = 0;
    for (Game g : games) {
        int size = g.getChips().size() - 1;
        int takeSize = g.getTakeAways().size();
        // Hledam nejvetsi odebiraci mnozinu krome her Nim
        if (g.getGameType() != 'n' && takeSize > takes)
            takes = takeSize;
        // priradim hodnoty sg-funkce poslednich chipu v radku
        sgvalues[i++] = g.getChips().get(size).getSg();
    }
    int [][] removes = new int[games.size()][takes];
    // zjistim pozici nejlevejsiho bitu v sume sg-hodnot her
    int leftmost = Integer.highestOneBit(xor(sgvalues));
    // pokud neni rovna 0
    if (leftmost != 0) {
        // prochazim radky hry
        for (int j = 0; j < games.size(); j++) {
            // ulozim si aktualni hodnotu sg-funkce
            int pom = sgvalues[j];
            int pos = 0;
            // prochazim vsechny chipy v radku hry
            for (int k = 0; k < games.get(j).getChips().size(); k++) {
                Game g = games.get(j);
                Chip ch = g.getChips().get(k);
                try {
                    // zkusim pridelit hodnotu sg-funkce aktualniho chipu
                    sgvalues[j] = g.getChips().get(k - 1).getSg();
                } catch (Exception e) {
                    // doslo k prekroceni velikosti pole
                    continue;
                }
                // jestlize je suma her po odebrani tohoto chipu rovna nule a je tah mozny
                if (xor(sgvalues) == 0 && g.isMovePossible(ch)) {
                    // pridam tento chip do optimalnich tahu
                    removes[j][pos++] = g.getChips().size() - ch.getNumber();
                }
            }
            // po projdeni radku hry vratim puvodni hodnotu sg-funkce
            sgvalues[j] = pom;
        }
    }
    return removes;
}
```

Výpis 3: Metoda pro výpočet optimálních tahů z dané pozice

Tato metoda je dále použita také při tahu počítače v rámci metody `computerPlay`. Protože z některých pozic existuje větší množství optimálních tahů, pro tah počítače je žádoucí, aby byl jeden z nich vybrán. Proto vznikla metoda `randomFromBest`, která vybírá

náhodný tah z množiny optimálních tahů. Metoda `getChipsArrayToRemove` je použita také při zobrazení informací o optimální strategii. Při inicializaci hry v rámci metody `init` se nastavují počáteční hodnoty řetězců, pomoci nichž je zobrazen optimální tah. Rovněž při obnovování hry metodou `repaint` pomocí metody `setStrategyLabels`, která nastavuje textové nápovědy u jednotlivých řádků součtové hry, je výpočet hodnot zobrazených těmito nápovědami realizován prostřednictvím metody `getChipsArrayToRemove`.

Pro realizaci tahu počítače bylo opět nutné implementovat metodu pro náhodný tah. `RandomMove` vrací pole dvou celočíselných hodnot, kde první z nich je náhodně vybraný řádek hry a druhá z nich je náhodně vybraný počet chipů, který bude z tohoto řádku odebrán. Jinak zůstal princip metody `computerPlay` z hlediska obtížnosti počítače zachován.

V rámci metody `repaint`, která je volána při vykonání každého tahu, jsou podobně jako u třídy `Game` obnoveny informace o jednotlivých podhrách a odebírací množiny těchto her. Na rozdíl od třídy `Game` je však nutné řádky součtové hry procházet v cyklu, kvůli jejich většímu počtu. Pomocí tohoto cyklu je také zjištěno, zda je hra již ukončena.

5.4.4 Nim

Zpracování logického základu hry Nim je zabudováno do třídy `Nim`. Tato třída dědí ze třídy `SumGame`, jedná se tedy opět o komponentu v rámci GUI. Tvoří ji určitý počet objektů, tedy konkrétních instancí třídy `Game`. Pro lepší manipulaci jsou tyto objekty opět uloženy pomocí datové kolekce `List`. Jejich grafická reprezentace ve výsledné komponentě je, stejně jako u součtové hry, ve formě řádků, neboli horizontálních layoutů, které dohromady tvoří `VerticalLayout`. Barevná reprezentace se od součtových her neliší.

```
public int getNimSum() {
    int nimSum = 0;
    for (Game g : games) {//games – List obsahujici vsechny podhry hry Nim
        nimSum ^= g.getChips().size() – 1;
    }
    return nimSum;
}
```

Výpis 4: Metoda pro výpočet nimsum

Metody třídy `Nim` by se daly opět rozdělit na dvě skupiny, kde první skupina se zabývá informacemi důležitými pro výukové účely aplikace a druhá skupina zajišťuje, že je možné si hru Nim na serveru zahrát. V první skupině figuruje jako jedna z nejdůležitějších metoda `getNimSum` (viz Výpis 4, str. 28), která vrací hodnotu nim-sumy za pomocí operace `xor`. Některé metody jsou podbné jako u třídy `Game`, avšak z různých důvodů bylo nezbytné, aby se mnoho z nich v implementaci lišilo. Protože je třída `Nim` potomkem třídy `SumGame`, dědí také všechny její metody. V některých případech však bylo nutné implementaci těchto metod změnit, což ve třídě `Nim` indikuje návěští `@Override` u příslušné metody.

Změny v implementaci na rozdíl od jednoduché odebírací hry vznikly především kvůli faktu, že hra Nim se skládá z několika podher. Použití jednoduchého cyklu pro procházení jednotlivých chipů se tedy zkomplikovalo a bylo nutné použít dva vnořené cykly.

Tento princip jsem musel použít jak při konstrukci samotné hry (v konstruktoru `Nim`), tak například později při zjišťování optimálního tahu pomocí metody `getChipsToRemove`. Také pro uložení tahu pro jeho další použití, například při realizaci tahu počítače, nebo při zobrazení optimální strategie, bylo třeba použít pole typu `int`. Tah se totiž skládá ze dvou parametrů, a to řádku hry, ze kterého se odebírá a počtu chipů k odebrání.

Metoda zprostředkovávající náhodný tah je odlišná ve srovnání s metodou třídy `SumGame`, bylo tedy nutné ji při implementaci překrýt. Používá pomocnou metodu `randomNumber`, která vrací náhodné číslo typu `int` z rozsahu od nuly po maximum specifikované v parametru. Návratová hodnota metody `randomMove` je stejná jako u třídy `SumGame` (viz Výpis 5, str. 29).

```
public int [] randomMove() {
    int [] move = new int[2];
    //vyberu nahodny radek hry Nim
    int game = randomNumber(numberOfGames);
    //pokud je tento radek prazdny, vybiram tak dlouho, dokud nenajdu neprazdny
    while (games.get(game - 1).getChips().size() <= 1) {
        game = randomNumber(numberOfGames);
    }
    //vyberu nahodny pocet chipu, ktery bude z radku odebran
    int chip = randomNumber(games.get(game - 1).getChips().size() - 1);
    move[0] = game - 1;
    move[1] = chip;
    return move;
}
```

Výpis 5: Metoda pro zprostředkování náhodného tahu

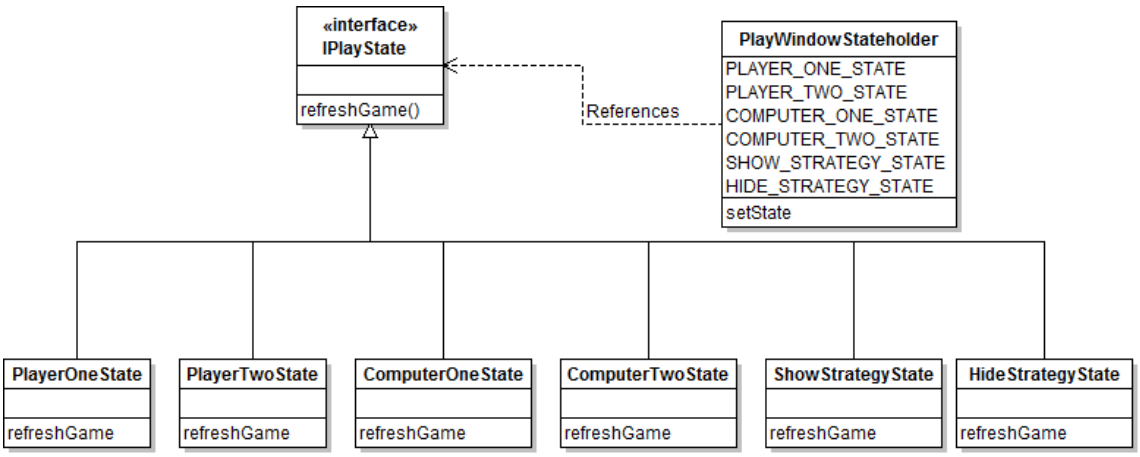
Rovněž příprava dat k zobrazení informací o optimální strategii si vyžádala odlišnou implementaci kvůli struktuře hry Nim. Pro nastavení řetězců zobrazených u jednotlivých her jako nápověda optimálního tahu byla použita metoda `setStrategyLabels`, která používá výše zmíněnou metodu `getChipsToRemove`.

Jedna z důležitých metod, která byla použita pro zjištění, zda je hra již ukončena, je metoda `getNumberOfAllChips`, jejíž návratová hodnota je celkový počet chipů ve hře Nim. Metody, které si svou podobnost se svými příbuznými ve třídě `Game` zachovaly, jsou zejména `computerPlay`, která zajišťuje tah počítače a metoda `repaint` pro překreslení hry. Musely však být opět překryty pomocí návěští `@Override`, kvůli jejich odlišnosti s původní implementací v rodičovské třídě `SumGame`.

5.5 Uživatelské rozhraní

V rámci hierarchické struktury projektu se uživatelské rozhraní aplikace nachází v balíku `gui` (viz Obrázek 4, str. 19). Dělí se dále na balíky `comb`, `nim` a `sum`, které odpovídají jednotlivým typům her, tedy odebírací hry, hra Nim a součtové hry. Každý tento balík dále obsahuje balíky `action` a `state`, které zahrnují třídy akcí a na ně navazující stavy. Třídy realizující zobrazení jednotlivých her jako stránky aplikace jsou `PlayGame`, `PlayNim` a `PlaySumGame`. Každá tato třída je potomkem `VerticalLayoutu`, což zařizuje, že stránka bude uspořádána vertikálně.

5.5.1 Akce a stavy



Obrázek 10: Třídní diagram stavů třídy `PlayGame`

Díky třídám akcí, které fungují jako listenery na podněty, které jsou vyvolány uživatelem prostřednictvím GUI aplikace, je možné měnit stavy, ve kterých se aplikace nachází. Tyto stavy jsou reprezentovány jednotlivými třídami stavů. Všechny třídy akcí vždy dědí z abstraktní třídy, která je potomkem třídy `ClickListener`, je tedy důležité, aby každá akce překrývala metodu pro kliknutí (`buttonClick`), ať už se jedná o tlačítko, nebo jiný ovládací prvek, který reakci pomocí této metody podporuje. Díky překrytí této metody je možné změnit stav příslušné zobrazené třídy. Implementace stavů je založena na návrhovém vzoru stav, všechny třídy stavů tedy implementují příslušné rozhraní. Změna stavu je realizována metodou `setState` třídy `StateHolder`, která zavolá metodu `refreshGame` příslušného stavu, čímž se aplikace aktualizuje do požadovaného stavu. Obrázek 10 (str. 30) dokresluje vztahy mezi třídami stavů v rámci balíku `comb` implementovanými pomocí tohoto návrhového vzoru.

Obrázek 11: Změna zobrazení při volbě odlišného typu hry

Pomocí akcí a stavů je realizována téměř každá změna zobrazení uživatelského rozhraní aplikace, pokud se nejedná o triviální změnu, jako například při výběru typu hry u odebíracích her. V tomto případě je pro změnu stavu, tedy zablokování či odblokování seznamu pro výběr odebírací množiny, použit inline listener typu `ValueChangeListener`. Funkčnost je znázorněna na obrázku 11 (str. 30). Podobně je tomu tak při dalších jednoduchých změnách v GUI.

Naopak veškeré provedení všech tahů, jak počítače, tak hráče, je realizováno pomocí tříd akcí a stavů. Rovněž zobrazení a skrytí informací o optimální strategii používá tento způsob implementace, spolu se složitějšími úkony, které jsou aktivovány buď tlačítkem, nebo změnou stavu některého jiného uživatelského vstupu. Tímto způsobem je například implementováno přídávání a odebírání her v rámci součtových her, ale také obnovování ovládacích prvků pro výběr chipů k odebrání. Následující úryvek kódu ilustruje implementaci pomocí návrhového vzoru state.

```
//rozhrani, ktere stavy baliku comb implementuji
public interface IPlayState {
    public void refreshGame(PlayGame playWindow);
}
// trida , ktera realizuje zmenu stavu
public class PlayWindowStateHolder {
    //vycet stavu, ktere budou implementovat rozhrani
    public static IPlayState PLAYER_ONE_STATE = new PlayerOneState();
    public static IPlayState PLAYER_TWO_STATE = new PlayerTwoState();
    public static IPlayState COMPUTER_ONE_STATE = new ComputerOneState();
    public static IPlayState COMPUTER_TWO_STATE = new ComputerTwoState();
    public static IPlayState SHOW_STRATEGY_STATE = new ShowStrategyState();
    public static IPlayState HIDE_STRATEGY_STATE = new HideStrategyState();
    //metoda pro nastaveni stavu
    public static void setState(IPlayState playState, PlayGame playWindow) {
        playState.refreshGame(playWindow);
    }
}
// priklad konkretniho stavu - pro schovani zobrazene strategie
public class HideStrategyState implements IPlayState {
    // prekryti (implementace) metody pro aktualizaci aplikace
    @Override
    public void refreshGame(PlayGame playWindow) {
        playWindow.getStrategy().setVisible(false);
        playWindow.getGame().getBestMoveLabel().setVisible(false);
        playWindow.getStrategyLabel().setVisible(false);
        playWindow.getInstructionLabel().setVisible(false);
        playWindow.getHelpLabel().setVisible(false);
        playWindow.getShowStrategyBtn().setCaption("Ukaz strategii");
        playWindow.getShowStrategyBtn().addListener(new ShowStrategyAction(playWindow));
    }
}
// trida akce, ktera realizuje listener na tlacitku , pomoci ktereho se schova zobrazena strategie
public class HideStrategyAction extends AbstractPlayAction {
    //konstruktor tridy
    public HideStrategyAction(PlayGame playWindow) {
        super(playWindow);
```

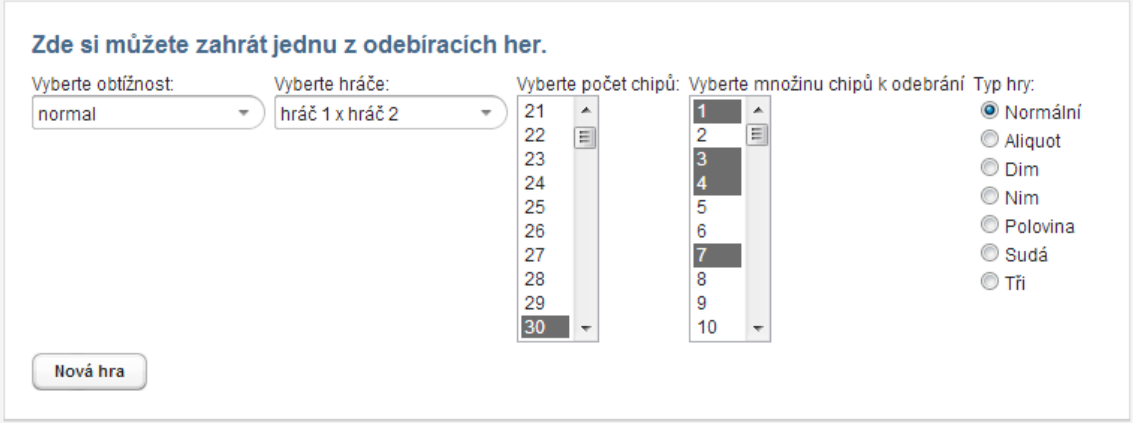
```
        }
        // prekryti  metody, nastavi stav HIDE_STRATEGY_STATE
        @Override
        public void buttonClick(ClickEvent event) {
            PlayWindowStateHolder.setState(
                    PlayWindowStateHolder.HIDE_STRATEGY_STATE, super.playWindow);
        }

    }
```

Výpis 6: Implementace akcí a stavů

5.5.2 Odebírací hry

Stránku na serveru, kde je možné si zahrát různé typy jednoduchých odebíracích her, reprezentuje třída `PlayGame`. V rámci této třídy stránku tvoří dva hlavní vertikální layouty. Jsou jimi `mainLayout`, který umožňuje zobrazení nadpisu stránky, hlavního nastavení hry a tlačítka pro spuštění hry (viz Obrázek 12, str. 32), a `gameLayout`, který zobrazuje vše potřebné ke hraní hry (viz Obrázek 13, str. 33).



Obrázek 12: Nastavení odebíracích her - `mainLayout`

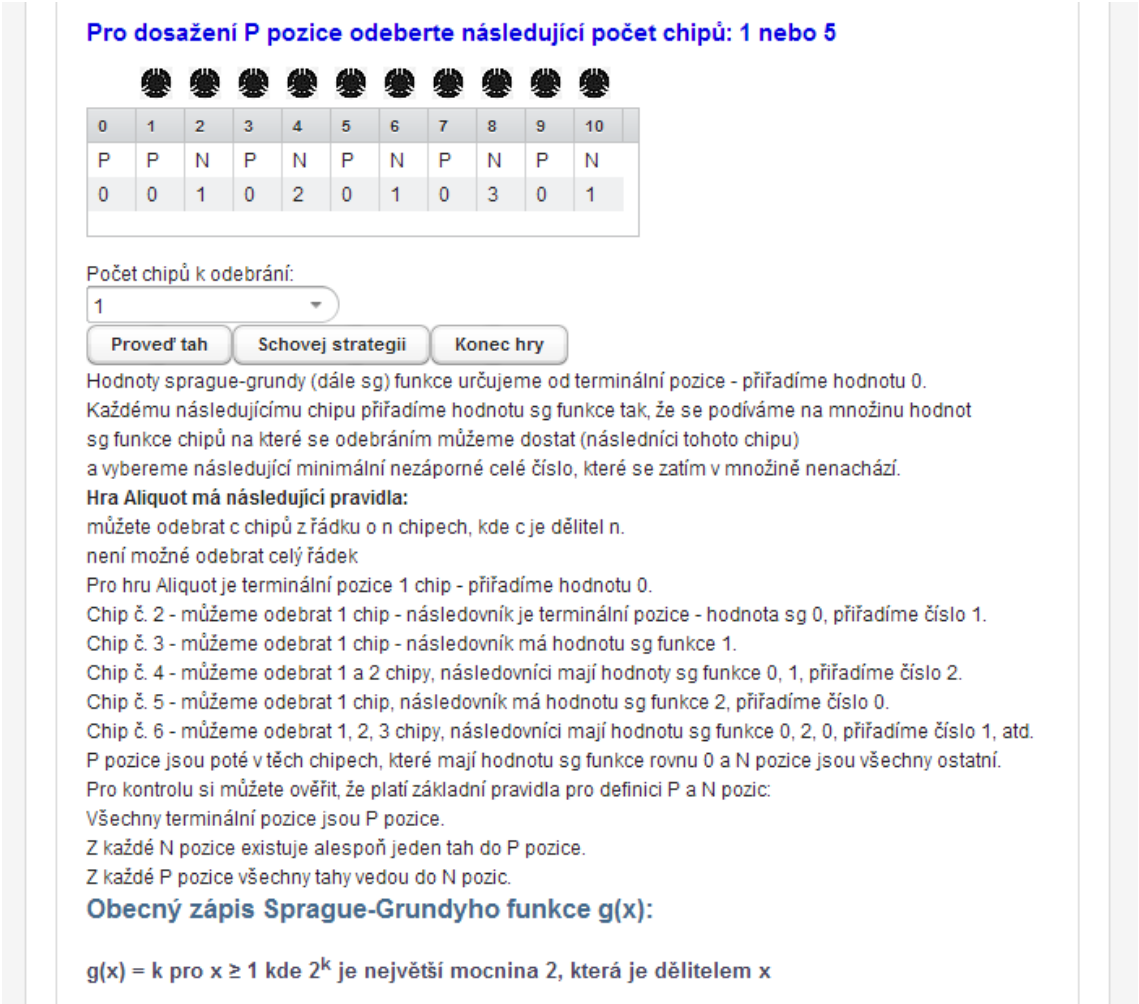
Při pohledu na hlavní layout si můžeme všimnout použitých komponent. První z nich je nadpis, který je zobrazen pomocí komponenty `Label` s příslušným stylem z tématu aplikace. Další dvě komponenty, umožňující volbu obtížnosti hry v případě hry proti počítači a volbu hráčů jsou realizovány `ComboBoxy`. Tyto objekty umožňují snadný výběr z menší skupiny možností. Další dva ovládací prvky jsou typu `ListSelect` kvůlí větší množině možností (počáteční počet chipů). Rovněž umožňují výběr více prvků z nabídky, pokud je tato možnost vyžadována, což je potřebné u výběru odebírací množiny. Poslední použitou komponentou je `OptionGroup`, která umožňuje v použité modifikaci výběr pouze jednoho prvku z nabízených možností typů her. Tyto ovládací prvky společně tvoří `horizontalLayout`, jsou tedy uspořádány horizontálně. Tlačítko „Nová hra“ umožňující spuštění hry je poslední součástí komponenty `mainLayout`. Spuštění hry

Obrázek 13: Zobrazení odebírací hry - `gameLayout`

je realizováno jednoduchou výměnou výše zmíněných dvou layoutů tím způsobem, že `mainLayout` je zneviditelněn a `gameLayout` je zviditelněn.

Layout pro hru tvoří dvě hlavní komponenty. První z nich je nadpis, opět realizovaný `Labelem` a tou zbývající je `Panel` obsahující vše potřebné pro zobrazení hry. Součástí tohoto `Panelu` je popis stavu hry, samotná hra a ovládací prvky hry, tedy `ComboBox` pro výběr počtu chipů k odebrání a tlačítka pro provedení tahu, zobrazení strategie a ukončení hry. Po kliknutí na tlačítko „Ukaž strategii“ jsou zobrazeny informace pro výukové účely, tedy optimální tah z dané pozice, tabulka s hodnotami SG-funkce a P-N pozicemi, obecný zápis SG-funkce a popis, jakým způsobem je možné se k těmto informacím dobrat. Veškeré textové informace jsou zobrazeny pomocí komponent `Label`. Většina z nich je použita v módu `Label.CONTENT_XHTML`, který usnadňuje formátování textu pomocí HTML tagů. Tyto komponenty jsou uspořádány pod sebou v `Panelu` pod zobrazenou hrou (viz Obrázek 14, str. 34). Pro usnadnění změny textu je většina textových informací definována jako instanční proměnné typu String, popřípadě pole Stringů v rámci deklarace privátních proměnných třídy.

Podstata hraní hry je realizována pomocí tlačítka „Proveď tah“, respektive „Proveď tah počítače“, pomocí kterých je spuštěna příslušná akce a hra je aktualizována do požadovaného stavu. Konkrétně například pro hru hráče proti počítači je tahem hráče aktivována akce `PlayerOneAction`, která hru přepne do stavu `PlayerOneState`. Vzhled hry se upraví pro provedení tahu počítače a po kliknutí na tlačítko „Proveď tah počítače“ se aktivuje akce `ComputerTwoAction` a hra se aktualizuje do stavu `ComputerTwoState`. Takovýmto způsobem probíhá střídání tahů jednotlivých hráčů. Tím, že se hra dostane do nějakého stavu, se aktivuje metoda `refreshGame` třídy tohoto stavu. Provedou se instrukce nezbytné k tomu, aby po provedení tahu byla zachována konzistence hry, tedy



Obrázek 14: Zobrazení informací o hře, optimální strategie

musí se odebrat příslušný počet chipů, aktualizovat odebírací množina a prostředí aplikace se musí změnit v závislosti na stavu hry a typu hráčů. Pro správnou funkčnost odebírání žetonů ze hry je nutné kontrolovat také množinu počtů žetonů, které je možné odebrat, protože u některých typů her se tato množina mění. Při každém provedeném tahu je tedy nutné znovu naplnit i `ComboBox`, který slouží k výběru počtu chipů k odebrání, což je realizováno metodou `fillChipsComboBox`. Hra je ukončena, jakmile je dosaženo terminální pozice. V tom případě dojde k zobrazení hlášení o konci hry a hráči, který vyhrál, v podobě notifikace (viz Obrázek 15, str. 35). Zároveň se objeví hlavní layout s ovládacími prvky pro nastavení nové hry a layout se hrou je zablokován. Stejný stav vyjma notifikace nastane po stisknutí tlačítka „Konec hry“

Obrázek 15: Notifikace o ukončení hry

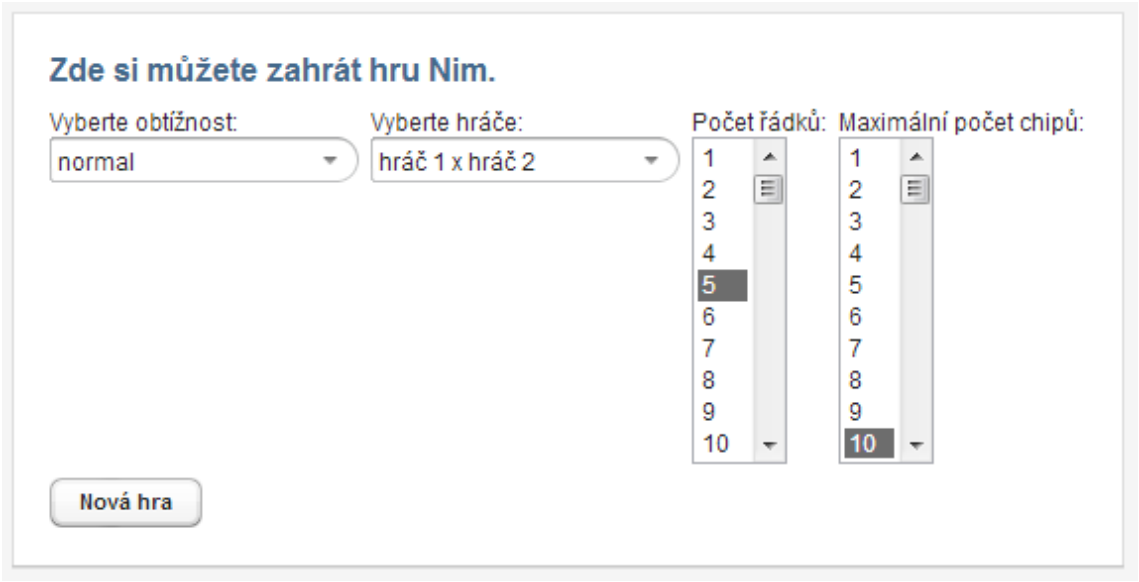
5.5.3 Hra Nim

Zobrazení rozhraní pro hraní hry Nim je zakomponováno do třídy `PlayNim`. Tato třída opět obsahuje dva základní layouty, a to `mainLayout` a `gameLayout`. Zobrazení hry Nim je založeno na stejném principu jako u odebíracích her, tedy hlavní layout v sobě zahrnuje nadpis, horizontální layout s ovládacími prvky pro nastavení hry a tlačítko pro spuštění hry. Layout pro hru pak zobrazuje samotnou hru Nim a vše co k ní patří. Tyto principy byly použity pro zachování konzistence zobrazení uživatelského prostředí a kvůli intuitivitě ovládání aplikace.

Ovládací prvky pro nastavení hry Nim se od odebíracích her téměř neliší (viz Obrázek 16, str. 36). Opět byly použity dva `ComboBoxy` se zachováním funkčnosti i pozice, které jsou následovány dvěma `ListSelecty`. První z nich nastavuje, kolik řádků bude hra Nim mít, a pomocí druhého můžeme vybrat, kolik chipů bude jeden řádek hry maximálně obsahovat. Chybí zde výběr typu hry, protože se jedná vždy o hru Nim. Tlačítko „Nová hra“ vytvoří instanci hry Nim a podobně jako u odebíracích her se zobrazí layout se hrou na úkor hlavního layoutu s ovládacími prvky pomocí metody `setVisible(true/false)` příslušných komponent (viz Obrázek 17, str. 37).

Pro ovládání hry byly opět použity 3 tlačítka. Tlačítko pro provedení tahu, zobrazení (respektive schování) strategie a tlačítko pro ukončení hry. Bylo nutné přidat `ComboBox` pro výběr řádku, ze kterého se bude odebírat. Pro správnou funkčnost byl naimplementován kromě stavů pro všechny hráče také stav pro změnu řádku v tomto `ComboBoxu`. Jakmile je vybrán jeden z řádků hry Nim, `ComboBox` pro výběr počtu chipů k odebrání je naplněn odpovídajícími hodnotami (odebírací množinou pro vybranou hru). Je tudíž zajištěno, že bude odebrán legální počet chipů (tah bude uskutečnitelný).

Při zobrazení strategie je vypuštěna tabulka s hodnotami SG-funkce a P nebo N pozicemi, protože pro hru Nim nemá smysl (viz Obrázek 17, str. 37). Naopak pro každý



Obrázek 16: Ovládací prvky nastavení hry Nim

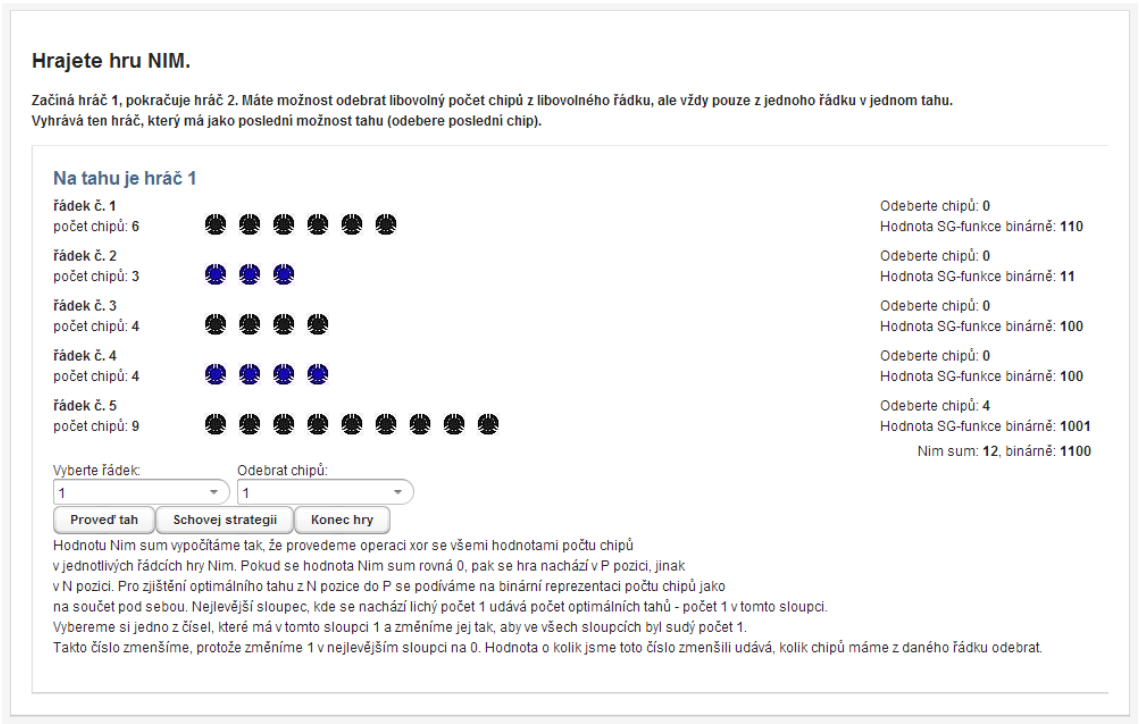
řádek hry je zobrazena hodnota počtu chipů, kterou je nutno odebrat pro dosažení P-pozice, případně 0, pokud této pozice nelze dosáhnout. Tato informace je následována binární hodnotou SG-funkce aktuální pozice, potřebnou pro výpočet nim-sumy, která je zobrazena pod těmito hodnotami. Veškeré textové informace jsou opět zobrazeny pomocí komponent typu `Label`. Tyto informaci jsou aktualizovány při provedení každého tahu, protože aktuální hodnoty SG-funkce a nim-sumy se vždy mění.

Konec hry nastává při dosažení terminální pozice jedním z hráčů a je realizován opět zobrazením notifikace o vítězi a ukončení hry. Poté je zobrazen layout s nastavením nové hry a herní layout je zablokován pomocí metody `setEnabled(false)`.

5.5.4 Součtové hry

Základem implementace zobrazení součtových her je třída `PlaySumGame`. Stejně jako dvě předchozí hry je zobrazení obsahu realizováno pomocí dvou layoutů, do kterých jsou naskládány všechny ostatní komponenty uživatelského rozhraní. Komponenta `mainLayout` opět zařizuje zobrazení nadpisu, ovládacích prvků pro nastavení součtové hry a tlačítko pro start nové hry. V herním layoutu se poté zobrazí nakonfigurovaná hra a příslušné rozhraní pro její ovládání.

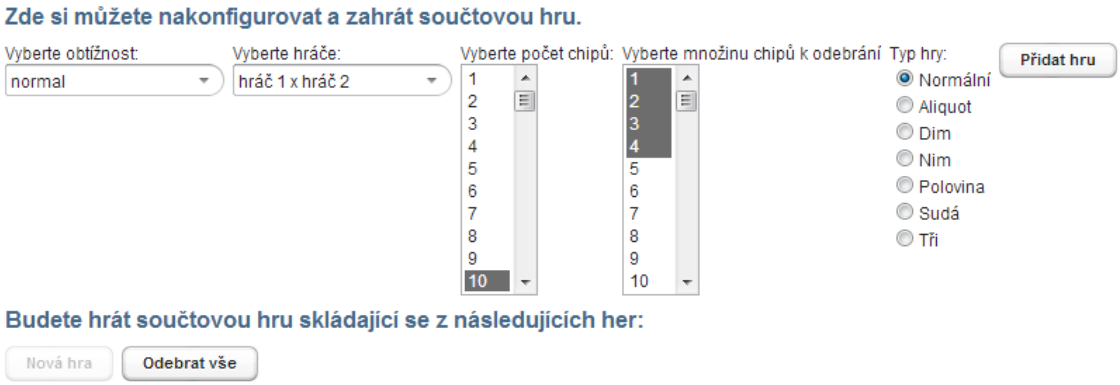
Nastavení součtové hry se na první pohled neliší od jednoduchých odebíracích her, důležitá jsou zde však nová tlačítka „Přidat hru“, „Odebrat hru“ a „Odebrat vše“, pomocí kterých je možné přidat libovolný počet her, odebrat kteroukoliv z nich, případně odebrat všechny (viz Obrázek 18, str. 38). Přidávání a odebírání her je realizováno opět pomocí akcí a jim odpovídajících stavů. `AddGameAction` zařizuje přidání hry prostřednictvím stavu `AddGameState`. Pomocí akce `RemoveGameAction` se aplikace dostane do stavu



Obrázek 17: Hra Nim - zobrazení informací o strategii

`RemoveGameState` a příslušná hra je odebrána. Tlačítko „Odebrat vše“ iniciuje akci `RemoveAllGamesAction` která vyvolá stav `RemoveAllGamesState` a všechny dříve přidané hry jsou odebrány.

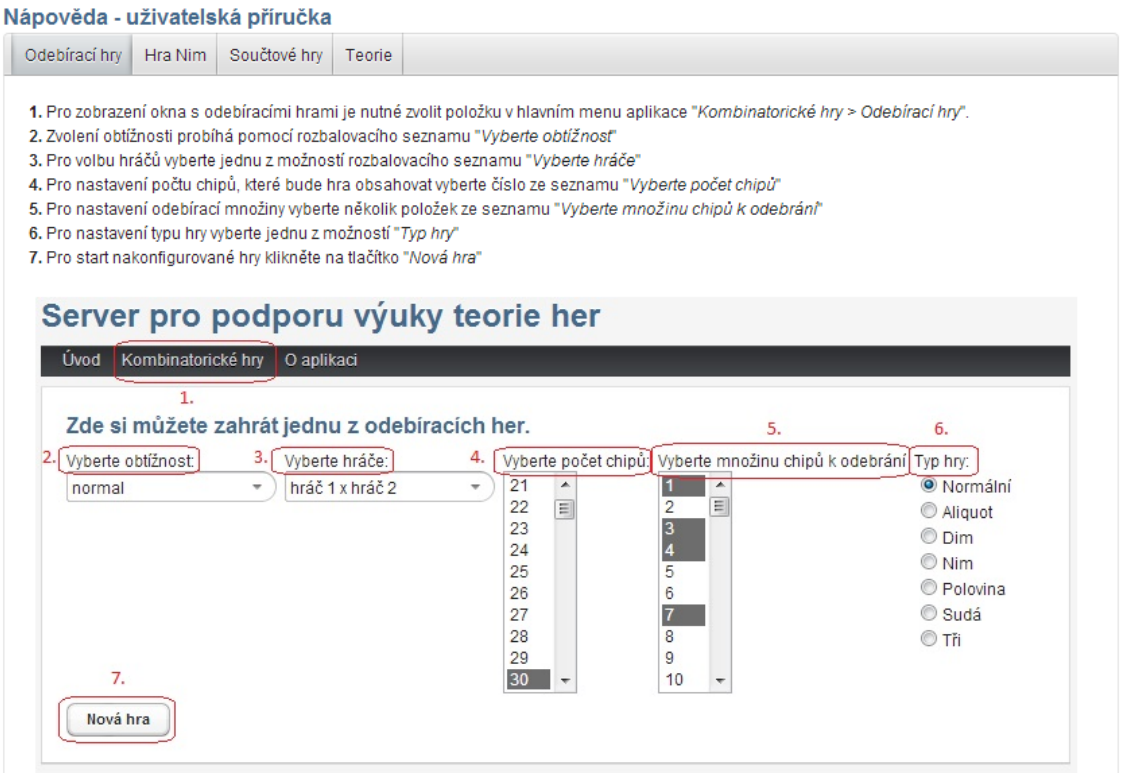
Spuštěním hry pomocí tlačítka „Nová hra“ je vytvořen objekt typu `SumGame` obsahující hry nakonfigurované způsobem popsaným výše. Zároveň je layout s ovládacími prvky zneviditelněn a zobrazena je součtová hra. Ovládání hry je postaveno na stejném principu jako u hry Nim. K zobrazeným informacím u jednotlivých her přibyla množina hodnot, které je možné z příslušné hry odebrat. Změnil se název rozbalovacího seznamu pro výběr hry, protože se vybírá hra, ze které se bude odebírat, místo řádku hry Nim. Při zobrazení informací o optimální strategii je naopak na rozdíl od hry Nim zobrazena u každé hry i tabulka s hodnotami SG-funkce a P-N-pozicemi, je totiž důležitá pro výpočet celkové sumy hry. Princip ukončení hry zůstal zachován jako u výše jmenovaných her.



Obrázek 18: Konfigurace součtové hry

5.5.5 Nápověda k aplikaci

Pro implementaci nápovědy aplikace byla použita třída v rámci balíku `gui`, jejíž název je `HelpWindow`. Zobrazení nápovědy je uvedeno nadpisem v podobě komponenty `Label`. Další součástí layoutu je komponenta `TabSheet`, která umožňuje členění obsahu pomocí záložek. V tomto případě se jedná o záložky podle obsahu jednotlivých stránek serveru, tedy Teorie, Odebírací hry, Hra Nim a Součtové hry. Každá záložka obsahuje vertikální layout, do kterého je členěn text a obrázky dohromady tvořící nápovědu k příslušné sekci. Pro umožnění jednoduché úpravy jsou texty definovány jako privátní proměnné třídy typu `String[]`, tedy pole řetězců. Zobrazení obrázků je realizováno opět pomocí komponenty `Embedded`, které je připojen příslušný zdrojový soubor pomocí komponent `ThemeResource`. Obrázky jsou tedy součástí tématu aplikace. Vzhled nápovědy ilustruje obrázek 19 (str. 39).



Obrázek 19: Zobrazení uživatelské příručky

6 Testování

Testování aplikace bylo díky faktu, že nebyl k dispozici tým testerů, ve skutečnosti součástí implementace. Při vývoji GUI aplikace bylo výstupem testování správné zobrazení jednotlivých komponent zarovnaných v příslušných layoutech. Ověřování správnosti probíhalo většinou v několika internetových prohlížečích, aby se zabránilo případné nekonzistenci zobrazení v závislosti na prostředí. Vždy po naimplementování určité sekce uživatelského rozhraní byla tato sekce pro kontrolu zobrazena a bylo otestováno správné zobrazení a zarovnání jednotlivých komponent.

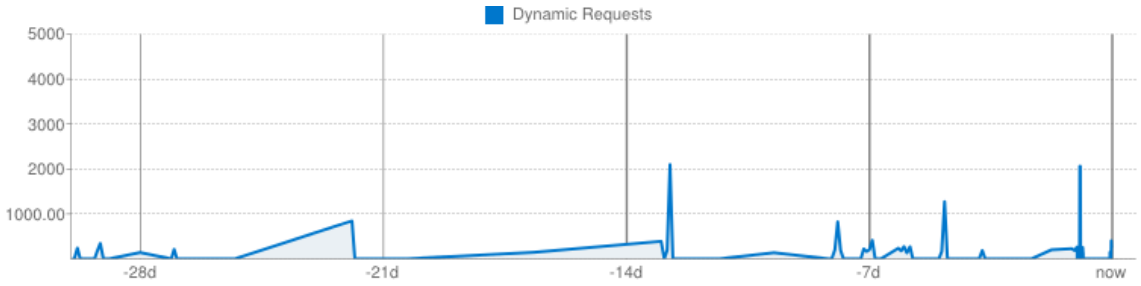
Jakmile došlo k implementaci funkčnosti aplikace, bylo třeba testovat správné chování při mnoha uživatelských vstupech. Tato činnost lze do určité míry automatizovat za pomocí vhodného software, já jsem se však omezil pouze na manuální testování aplikace. Oblast testování vždy závisela na tom, jaká část aplikace byla v danou chvíli dokončena, tedy zaměřoval jsem se na správnou funkčnost právě dokončeného kódu.

Při opravách nefunkčních nebo špatně fungujících částí aplikace bylo nutné po provedení zásahu otestovat jak opravenou část aplikace, tak části, které mohly být ovlivněny změnou v implementaci. V některých případech bylo tedy použito i regresní testování. Jelikož jsem používal pro uchování různých stavů implementace verzovací systém SVN, jakékoli chyby v implementaci zjištěné při testování, které vyžadovaly návrat k předchozí verzi, bylo možné tímto způsobem odstranit.

7 Nasazení

Protože je server pro podporu výuky teorie her webovou aplikací, jeho nasazení je možné provést na webový server. V rámci testování jsem používal lokální server Tomcat ve verzi 7.0. Umožňuje jednoduché spuštění aplikace a také ladění, což je při vývoji velmi důležité.

Také jsem vyzkoušel nasazení aplikace na server společnosti Google, protože tuto službu pod jménem *app engine* poskytuje na omezenou zkušební dobu zdarma. Tato služba je velice podobná reálnému nasazení aplikace. *App engine* navíc umožňuje různá nastavení aplikace. Jedním z nich je ovlivnění výkonnosti serveru, je tedy možné nasimulovat, jak se aplikace bude chovat vzhledem k použití různého množství systémových prostředků. Další vlastností, kterou tato služba zprostředkovává, je udržování mnoha statistik o aplikaci. Například je zde k dispozici graf zobrazující počet milisekund, které je potřeba ke zpracování požadavku v rozmezí 30 dní (viz Obrázek 20, str. 41).



Obrázek 20: Graf rychlosti zpracování požadavku

Pro nasazení aplikace je tedy za potřebí server s podporou Javy. Aplikace je nasazena na server v podobě WAR archivu jako servlet. Tento typ souboru je podtyp Java archivu (JAR), je tedy zakomprimován jako ZIP soubor se speciální strukturou obsahu. Nasazení aplikace je v podstatě přiřazení Java servletu na aplikační Java server, neboli servlet kontejner.

8 Závěr

Hlavním cílem diplomové práce byl vývoj výukového serveru pro Teorii her. K tomu bylo zapotřebí nastudovat teorii důležitou k pochopení daného tématu a všech potřebných souvislostí. Tato informační báze je vysvětlena v první části práce v kapitole Teorie her (str. 7). Primárními zdroji pro studium teorie her byly elektronický výukový dokument [2] a předmět Teorie her a modely rozhodování v podmínkách neurčitosti přednášený na Fakultě elektrotechniky a informatiky Vysoké školy báňské - Technické univerzity Ostrava. Po úspěšném absolvování tohoto předmětu jsem měl teoretický základ potřebný k navržení a implementaci výukového serveru.

Server pro podporu výuky Teorie her je webová aplikace implementovaná v jazyce Java postavená na komponentním frameworku Vaadin, který je určen pro vývoj GUI. Na serveru je možné si zahrát několik typů odebíracích her, mezi něž patří např. Aliquot, Dim a Nim. Server také nabízí možnost vytvořit si libovolnou součtovou hru, která se skládá z několika podher libovolného typu. Hraní her je umožněno jak proti počítači, který uplatňuje strategii podle zvolené obtížnosti, tak proti lidskému protivníkovi na stejném počítači. Kromě hraní her bylo jedním z hlavních cílů práce zobrazení optimální strategie k dané hře a informací, které jsou potřebné k jejímu získání. Optimální strategie je soubor informací důležitých pro vítězství hry. Jsou jimi hodnoty Sprague-Grundyovy funkce, respektive znalost, zda se hra nachází v P nebo N pozici, z čehož je možné se dozvědět tah vedoucí k výhře. Další informace, které jsou v aplikaci zobrazitelné, jsou obecné vzorce Spraguy Grundyovy funkce pro zvolenou hru a návody a principy, jak je možné se k optimální strategii dobrat.

Přínosem této práce bylo zdokonalení práce s programovacím jazykem Java a jeho frameworkem Vaadin, nové zkušenosti s některými pluginy v prostředí Eclipse, které usnadňují tvorbu UML diagramů. Dalším přínosem bylo naučit se používat některé návrhové vzory a smysluplně je využít v rámci projektu. Při příští implementaci bych na základě nabytých zkušeností sice navrhl architekturu aplikace mírně odlišně, ale použití návrhových vzorů bych zcela jistě zachoval. Pozitivní zkušeností také bylo zkušební nasazení na aplikační server společnosti Google, který má velké množství zajímavých funkcí.

V rozvoji aplikace jako výukového serveru je možné pokračovat, a to implementací dalších typů her, protože jich existuje velké množství. Z mého pohledu je tato výuková alternativa zajímavou změnou oproti studování rozsáhlých skript. Umožňuje vyzkoušet si nastudovanou teorii v praxi na konkrétních příkladech a navíc zprostředkovává učivo zábavnou formou.

9 Reference

[1] Wikipedia, The free encyclopedia, *Teorie her. In: Wikipedia: the free encyclopedia [online]. [cit. 2013-03-04]. Dostupné z: http://cs.wikipedia.org/wiki/Teorie_her*, San Francisco (CA): Wikimedia Foundation, 2001.

[2] FERGUSON, Thomas S., *Game Theory. elektronický výukový text [online]. [cit. 2013-03-04]. Dostupné z: http://www.math.ucla.edu/tom/math167.html,* 2005.

[3] GRÖNROOS, Marko, *Book of Vaadin. Vaadin [online]. [cit. 2013-03-13]. Dostupné z: https://vaadin.com/book*, VAADIN LTD, 2000 - 2013.

[4] VAADIN, ltd, *Thinking of U and I. Vaadin [online]. [cit. 2013-04-04]. Dostupné z: https://vaadin.com/home*, VAADIN LTD, 2012.

A Obsah CD

A.1 gts - aplikace Server pro podporu výuky teorie her

- .settings - nastavení projektu používaného v IDE Eclipse
- build - přeložené soubory tříd
- doc - dokumentace vygenerovaná pomocí programu javadoc
- src - zdrojové kódy aplikace
- WebContent - soubory webové aplikace
- .classpath - soubor důležitý pro nastavení projektu
- .project - projekt IDE Eclipse

A.2 doc - text diplomové práce

- tex - zdrojové soubory pro LaTeX a použité obrázky a diagramy
- gts.pdf - text diplomové práce